Carlos Eduardo da Silva Pereira

Carlos Eduardo da Silva Pereira **Middleware Genérico para Aplicações Web**

UMinho | 2009

Universidade do Minho
Escola de Engenharia

Carlos Eduardo da Silva Pereira

Middleware Genérico para Aplicações Web

Novembro de 2009

Universidade do Minho
Escola de Engenharia

Carlos Eduardo da Silva Pereira

Middleware Genérico para Aplicações Web

Mestrado em Informática

Trabalho efectuado sob a orientação do
Professor Doutor António Nestor Ribeiro

Novembro de 2009

Resumo

Cada vez mais, os Web Services possuem um papel de destaque como fontes de conteúdos para a construção de aplicações graças a uma elevada capacidade de reutilização a si inerentes. Este crescimento deu origem a aplicações, os *Mashups*, onde a informação provém unicamente deste tipo de fontes. Apesar disto, a sua utilização ainda tem subjacentes alguns problemas críticos na concepção e execução das aplicações. A elevada dependência das aplicações em apenas um fornecedor de serviços, pode, em caso de indisponibilidade deste, causar falhas na própria aplicação, reduzindo a sua efectividade. O processo de selecção do fornecedor a utilizar num determinado contexto obriga os programadores a um estudo aprofundado das características dos diversos serviços, de forma a encontrar o que melhor preenche as suas necessidades. Por sua vez, a inclusão na aplicação de conteúdos provenientes de vários serviços pode tornar-se um processo complexo devido a diferenças nas tecnologias de implementação de cada fornecedor. Deste modo, os tempos de desenvolvimento das aplicações tornam-se longos e por conseguinte mais dispendiosos.

Neste sentido, esta dissertação focalizou-se na concepção de um middleware genérico para aplicações Web, assente sobre um modelo de mediação de serviços baseado no modo de entrega definido por Software as a Service. O seu objectivo consiste em fornecer aos programadores uma forma simples e eficaz de utilizarem serviços na concepção das suas aplicações, através da uniformização dos processos de invocação e formatos de resposta dos diversos fornecedores, segundo o seu contexto, numa só API de serviço. Em adição, o modelo compreende políticas de melhor fornecedor assentes em aspectos relacionados com qualidade de serviço (QoS) de modo a existir uma eleição do fornecedor a utilizar, de acordo com as intenções dos programadores. De forma a demonstrar a validade da solução, foi construído um sistema de mediação real, tendo sido efectuada uma prova de conceito onde as vantagens provenientes da utilização do mesmo foram apresentadas.

Palavras-Chave: Web Services, Middleware, Mashup, Software as a Service (SaaS), Qualidade de Serviço, Service-Oriented Architecture (SOA), Aplicações Web, Mediação de Serviços

Abstract

Nowadays, the role of Web Services is taking the shape of information sources applied in the development of applications, mainly due to their underlying reuse capabilities. This growth gave birth to applications, like Mashups, which are completely based on this type of information sources. However, several critical problems related with conception and execution of applications still remain. The high dependency of applications on a single service provider, may, in case of service unavailability, cause failures on the very application, reducing its effectiveness. The process of finding the right provider for a determined context, implies a profound study about services' characteristics, by the programers, in order to find the one which better fulfills their needs. On the other hand, the inclusion of contents from several service providers, in the application, may become a complex process due to the different implementation technologies of each provider. This way, the time spent on the application development may become longer, raising the expensiveness of such a task.

In this context, this thesis focused on the conception of a generic middleware for Web Applications, based on a model for services mediation, constituted upon Software as a Service concepts. Its objective consists on providing, to programers, a simple and efficient way of using services in the development of their applications. This is done by standardizing the invocation processes and answer formats of the several providers based on their context, in a single service API. Moreover, the model embodies best provider policies, based on aspects related with the quality of service (QoS), in order to elect the provider to use according to programers' intentions. To show the validity of this solution, a real mediation system was developed and the advantages of its usage were presented in a concept proof.

Palavras-Chave: Web Services, Middleware, Mashup, Software as a Service (SaaS), Quality of Service, Service-Oriented Architecture (SOA), Web Applications, Service Mediation

Agradecimentos

Na realização deste trabalho, foram muitos aqueles que me incentivaram e ajudaram, e sem os quais a constituição deste seria muito provavelmente apenas uma miragem.

Em primeiro lugar, tenho que agradecer aos meus pais, fonte inesgotável de apoio, por terem feito tudo ao seu alcance para que conseguisse que este objectivo fosse a bom porto. Sem eles, não tenho dúvida, tudo o que consegui até hoje seria impossível.

Agradeço ao Professor António Nestor Ribeiro, por toda a ajuda disponibilizada ao longo destes meses.

Agradeço em especial ao João Granja e ao Nuno Oliveira, por serem dois amigos com quem tenho a certeza que posso contar e cuja amizade prezo acima de todas as restantes.

Por fim, agradeço ao Artur, que para além de familiar é um grande amigo.

Conteúdo

Lista de Figuras

Acrónimos

A

AJAX Asynchronous Javascript And XML, p. 1.

API Application Programming Interface, p. 1.

ASP Application Service Provider, p. 2.

B

BPEL Business Process Execution Language, p. 25.

C

COM Component Object Model, p. 9.

CORBA Common Object Request Broker Architecture, p. 9.

CRM Customer Relationship Management, p. 25.

CSS Cascading Style Sheets, p. 41.

D

DCOM Distributed Component Object Model, p. 13.

DHTML Dynamic HyperText Markup Language, p. 40.

DSL Domain-Specific Language, p. 42.

E

`EAI` Enterprise Application Integration, p. 35.

`ERP` Enterprise Resource Planning, p. 25.

`ESB` Enterprise Service Bus, p. 27.

`EUP` End-User Programming, p. 21.

F

`FTP` File Transfer Protocol, p. 12.

H

`HCI` Human–Computer Interaction, p. 31.

`HTML` HyperText Markup Language, p. 10.

`HTTP` Hypertext Transfer Protocol, p. 10.

J

`JMS` Java Message Service, p. 27.

`JSON` JavaScript Object Notation, p. 12.

L

`LAN` Local Area Network, p. 36.

M

`MIME` Multipurpose Internet Mail Extensions, p. 12.

`MOEM` Multidimensional Object Exchange Model, p. 43.

P

`PDF` Portable Document Format, p. 22.

`PHP` Hypertext Preprocessor, p. 23.

Q

QoS Quality of Service, p. 44.

R

REST Representational State Transfer, p. 1.

RoR Ruby on Rails, p. 43.

RSS Really Simple Syndication, p. 23.

S

SaaS Software as a Service, p. 3.

SMTP Simple Mail Transfer Protocol, p. 12.

SOA Service Oriented Architecture, p. 3.

SOAP Simple Object Access Protocol, p. 1.

U

UDDI Universal Description Discovery and Integration, p. 1.

UML Unified Modeling Language, p. 35.

URI Uniform Resource Identifier, p. 14.

URL Uniform Resource Locator, p. 1.

W

W3C World Wide Web Consortium, p. 1.

WAN Wide Area Network, p. 36.

WSDL Web Service Definition Language, p. 1.

X

`XAML` Extensible Application Markup Language, p. 40.

`XML` eXtensible Markup Language, p. 1.

`XML-RPC` XML Remote Procedure Calls, p. 1.

`XPath` XML Path Language, p. 42.

`XSLT` EXtensible Stylesheet Language Transformations, p. 42.

Capítulo 1

Introdução

Nos últimos anos tem-se desenvolvido uma tendência no sentido de serviços online serem abertos ao exterior através da elaboração de *Application Programming Interfaces* (API) públicas. Estas APIs podem ser descritas como o elo de ligação entre as aplicações e os serviços online, mais conhecidos como Web Services [Gur04].

De acordo com a *World Wide Web Consortium* (W3C) [W3Cf], Web Services podem ser definidos como aplicações modulares, auto-contidas e auto-descritas. O W3C Working Draft Glossary [W3Ce] identifica-os como "um sistema de software identificável por um URL, cujas interfaces públicas e respectivas ligações são definidas por XML". Em suma, um Web Service retrata um serviço utilizado através da Web, com comunicações baseadas em mensagens XML, descrita por intermédio de uma API pública, e acessível por um URL.

A sua implementação retrata um modelo cliente-servidor, concebido através de tecnologias tais como WSDL/SOAP [Cer02], REST [Fie00], AJAX [Gar], feeds, ou até mesmo *XML Remote Procedure Calls* (XML-RPC) [US]. Por vezes são ainda utilizados repositórios centrais para as APIs (como o UDDI [OASb]), de forma a que os fornecedores possam expor os serviços criados e estes possam ser facilmente encontrados pelos clientes.

Para além das utilizações naturais, tais como fornecimento de conteúdos, os Web Services permitem ainda facilitar a integração entre aplicações, tornando-as, assim, menos dispendiosas. De acordo com o site Programmable Web[1] o número de APIs conhecidas disponíveis aumenta a um ritmo que ultrapassa as 30 por mês. Este crescimento no número de Web Services disponíveis promove o aparecimento de aplicações Web cujos conteúdos resultam da composição de conteúdos disponibilizados por terceiros [ZR08]. Estas aplicações, que têm como principal objectivo a integração de múltiplas fontes de informação, ou APIs, numa única interface, são designadas por *mashups* [MS08].

Como exemplo de mashup pode-se considerar uma aplicação que combina informa-

[1] apontador que mede a variedade de web services existentes

ção sobre o estado do tempo (recolhido por exemplo do YahooWeather) com um mapa (recolhido do GoogleMaps), fornecendo ao utilizador uma interface única que mostra o estado do tempo de acordo com a posição geográfica num mapa. Para tal, a aplicação recorre a dois Web Services para obter informação, que combina e disponibiliza segundo uma interface própria.

Talvez a principal diferença entre mashups e aplicações baseadas em componentes esteja no facto de que o desenvolvimento de mashups pretende normalmente resolver uma situação específica e de curta duração, com base em tecnologias Web. Desta forma, o objectivo dos mashups prende-se com a simplicidade, usabilidade e facilidade de acesso, factores que se tornam mais importantes que extensibilidade ou uma elevada completude de recursos [YBCD08]. Deste modo, os mashups tornaram-se por larga margem, o meio mais popular de criação de aplicações de curta duração, ou seja, focadas num problema muito específico e actual. Com o tempo médio de integração de uma aplicação a situar-se entre três a seis meses [Wat07], existe uma curta margem para retorno do investimento. A indústria necessita de soluções que possam ser construídas rapidamente e com baixos custos de desenvolvimento para solucionar problemas específicos e imediatos, e ainda assim serem consideradas dispensáveis (quando o mercado evoluir). Neste sentido, torna-se necessário que o processo de desenvolvimento deste tipo de aplicações preencha estes requisitos, ou seja, se torne simples e de rápida execução.

No entanto, o desenvolvimento de mashups ainda não reflecte na totalidade estes aspectos. Dificuldades inerentes à própria estrutura do mashup, como a necessidade de construção de interfaces para o utilizador, para além da necessidade de análise, agregação e manipulação da informação recolhida, aliada à própria dificuldade das linguagens de desenvolvimento necessárias, tornam estes tipo de tarefas difíceis para utilizadores sem elevadas competências a nível de desenvolvimento Web.

Normalmente, a arquitectura utilizada na concepção de um serviço Web utilizado pelos mashups é a arquitectura típica de um *Application Service Provider* (`ASP`). No entanto, este tipo de arquitectura engloba uma quantidade de problemas envolvendo aspectos como a própria disponibilidade do servidor aplicacional, falhas provocadas pelas próprias comunicações, limitações do servidor ou até mesmo indisponibilidade de prestação do serviço pelo `ASP`. De facto, a própria natureza dos mashups não favorece esta implementação devido à sua estrutura fortemente baseada em eventos. No fundo, os mashups são essencialmente aplicações reactivas, sensíveis a eventos de fontes de informação, tais como uma feed de notícias ou até uma interacção efectuada pelo utilizador. No entanto, estes problemas não são exclusivos aos mashups. De facto, aplicações Web que utilizem recursos dispersos sob a forma de Web Services padecem das mesmas dificuldades, afectando

gravemente o seu desempenho.

Um modelo alternativo e actual, pensado como forma de responder a estes problemas designa-se por *Software as a Service* (SaaS).

SaaS como Middleware Genérico

A origem de *Software as a Service* (SaaS) remonta a 1999 por Pearl Brereton [TBB03], quando este afirmou que o futuro do software não iria ser baseado no desenvolvimento de novas arquitecturas com base em objectos ou componentes, mas em algo radicalmente diferente: na forma como o software iria ser entregue ao utilizador.

Os defensores de SaaS acreditam que o software deve ser fornecido como um serviço. Logo, o próprio conceito de SaaS defende um modelo de entrega de software [LT08]. Este modelo de entrega essencialmente retira ao cliente deveres sobre o software. Ao executar o software do lado do fornecedor, este torna-se responsável por toda a sua implementação, assim como pela infra-estrutura. O fornecedor é ainda responsável pelo fornecimento do mesmo (a pedido), normalmente através de uma arquitectura via Internet ou Intranet [PH07].

Consequentemente, as funcionalidades do software passarão a ser fornecidas através de um conjunto de serviços distribuídos que possam ser configurados e integrados no início da própria transacção. Deste modo, problemas relacionados com uso, instalação e configuração, e até mesmo evolução do software podem mais facilmente ser resolvidos. Dado que aspectos relacionados com infraestrutura e manutenção do software passam para o fornecedor, o cliente obtém uma diminuição no custo do software. Em adição, uma maior especialização do software é conseguida, podendo constituir uma mais valia para pequenas e médias empresas.

Para além de SaaS, a utilização de serviços web deu origem a novas arquitecturas focalizadas na sua utilização.*Service Oriented Architecture* (SOA) define-se como um modelo arquitectural distribuído, em que as aplicações são encapsuladas como serviços, com um sistema de comunicação próprio acessível por rede. Estabelecendo uma comparação entre SaaS e SOA, a principal diferença centra-se com o facto de que em SOA, os serviços são disponibilizados como componentes de maneira a serem implementados em novos sistemas de software [IHJ+07]. Em SaaS, os serviços são disponibilizados ao cliente que os utiliza a pedido na arquitectura do fornecedor. Apesar da distinção, a verdade é que ambos se complementam muito bem, sendo possível a utilização de SaaS para obtenção de componentes para SOA ou até mesmo utilizar SOA para obter SaaS. Isto é facilitado pelo facto de que quer SaaS, quer SOA, utilizarem tecnologias de Web Services que apesar de não

serem obrigatórias, são sem dúvida a melhor opção para a sua implementação [LZV08].

Numa definição quase ideal da implementação de SaaS, os serviços seriam dinamicamente compostos quando necessário através da ligação com outros de menor dimensão. A solução idealizada seria talvez a obtenção destes mesmos serviços dinamicamente através da composição de serviços de menor escala, com base em descrições elaboradas pelo utilizador sobre as tarefas que pretende resolver.

A nível arquitectural, SaaS distingue-se do procedimento normal em ASPs por possuir um middleware intermédio, que retira a necessidade do utilizador contactar directamente com o Web Service. Assim, a aplicação cliente apenas comunica com o middleware, e este fica responsável pela obtenção dos conteúdos desejados. Assente sobre os princípios de SaaS, esta tese propõe-se fundamentalmente simplificar a construção e execução de aplicações Web, como mashups, que utilizem serviços Web dispersos, através da criação de um middleware genérico para serviços baseado num modelo arquitectural.

Pretende-se que através da utilização de um sistema intermediário, problemas como a indisponibilidade de um fornecedor de serviço possam ser minorados através da capacidade do mediador em invocar outro fornecedor capaz de disponibilizar o mesmo tipo de conteúdos. Neste sentido, pretende-se que uma camada de eleição de serviços web seja integrada no middleware, capaz de disponibilizar flexibilidade ao nível do fornecedor de serviço a eleger, algo inexistente no modelo cliente-servidor habitual.

Outro problema habitual derivado da implementação de serviços Web provém da necessidade dos programadores estudarem diversas APIs de serviço e consequentemente necessitarem de implementar nas suas aplicações, mecanismos diferentes de forma a obter os conteúdos pretendidos de cada fornecedor. Como consequência, isto provoca um aumento significativo nos tempos de desenvolvimento da aplicação e consequentemente nos seus custos de produção. Através da introdução de um intermediário entre os clientes e os fornecedores, pretende-se que exista uma uniformização nos formatos de invocação dos serviços assim como nos resultados obtidos pelos mesmos. Deste modo, as aplicações interagem apenas com o middleware, ficando a cargo deste todo o processo de invocação dos diversos fornecedores e suas APIs, facilitando assim a concepção das aplicações.

Adicionalmente, na concepção de aplicações Web que utilizam serviços Web, é por vezes difícil aos programadores a selecção das APIs a utilizar. Por exemplo, um programador pode pretender um serviço relativamente simples, ou seja, com poucas funcionalidades, mas que possua uma elevada disponibilidade. No entanto, para o programador, a percepção de qual o fornecedor que oferece melhor disponibilidade pode ser um processo algo complexo. Neste sentido, pretende-se que o middleware constituído obtenha capacidades de análise à qualidade de serviço (*Quality of Service*) dos fornecedores. Assim, com base

em aspectos relacionados com a qualidade de serviço, a aplicação cliente pode informar o middleware de qual o requisito mais importante na sua óptica, permitindo assim que lhe seja fornecido o melhor serviço para o objectivo que esta pretende alcançar.

De facto, o cálculo da disponibilidade dos fornecedores de serviço, a ser efectuado com base em termos estatísticos pelo middleware, resolve assim um dos principais problemas existentes na arquitectura de `ASP`s e por conseguinte nas aplicações Web por si suportadas. Outros aspectos de qualidade de serviço podem-se dividir em aspectos relacionados com *runtime* tais como desempenho, tempo de resposta ou disponibilidade, aspectos transaccionais, como por exemplo integridade, questões de custos e configuracionais, ou até mesmo aspectos relacionados com segurança, como autenticação, cifragem ou confidencialidade. Uma possível implementação destes aspectos passa pela criação de métricas específicas para o efeito adicionadas ao middleware. Outra solução poderia passar pela adição de uma componente semântica às descrições `WSDL` dos serviços, o que no entanto implicaria a adopção destes pelos seus proprietários.

Uma alternativa à abordagem centrada na qualidade de serviço, passaria pela implementação de um directório de serviços, com recurso a um grafo ligado, catalogado por contexto, ou seja, por cada categoria de serviço (exemplo: fotografia), um contexto seria definido. Desta forma, a procura pelos serviços é simplificada, com base nas informações fornecidas pelo cliente [DVV03]. No entanto, este tipo de soluções não tem por base a eleição do melhor serviço, mas do mais compatível. Assim, e apesar de um serviço poder encontrar-se offline 12 horas por dia, se no processo de *matching* se tratar do mais compatível, será esse o fornecido.

A principal diferença entre contexto e qualidade de serviço prende-se com o facto de um contexto se tratar de uma característica do pedido, tal como possuir uma determinada extensão ou tamanho, enquanto em termos de qualidade de serviço se tratar de um requisito funcional, ou não funcional, tal como latência ou largura de banda [Ran03]. Apesar disto, a implementação dos dois simultaneamente, embora mais exigente pela parte do servidor, poderia garantir um maior rigor na eleição do melhor serviço com base nos requisitos fornecidos pelo cliente. Com isto, o middleware a criar adquire uma capacidade de proporcionar aos programadores de aplicações tanto informações comportamentais dos serviços como características funcionais dos mesmos, que, para além de retirar ao programador a necessidade de análise do fornecedor que melhor lhe convém, simplifica o processo de concepção das aplicações Web.

1.1 Objectivos

Enquadrados nos objectivos gerais delineados na secção anterior, os objectivos específicos subjacentes a este trabalho de mestrado e consequente dissertação são os seguintes:

- Estudo do conceito de "software as a service" e da criação de aplicações com recurso a serviços dispersos e acessíveis via `APIs` públicas.
- Criação de um middleware que encapsule diversos serviços externos e possibilite às aplicações web a serem desenvolvidas o acesso a esses serviços de uma forma uniforme e independente do fornecedor.
- Criação de mecanismos adjacentes ao middleware por forma a permitir que o serviço disponibilizado seja o que contenha melhor qualidade de serviço.
- Criação de uma aplicação-exemplo que demonstre a validade do middleware que se venha a desenvolver. Esta aplicação deverá integrar diferentes tipos de serviços, constituindo-se como um *mashup*.

1.2 Organização do Documento

Para além deste capítulo introdutório, esta dissertação é composta por 6 capítulos, resumidos de seguida.

Nos capítulos 2 e 3 são abordados diversos assuntos relativos à problemática trabalhada nesta dissertação. No capítulo 2 é elaborada uma introdução à temática dos Web Services e é feita uma análise comparativa entre as tecnologias apresentadas. São explicados os fundamentos presentes na concepção de aplicações compostas unicamente por serviços Web, os Mashups. Como consequência da utilização de serviços Web, apresentam-se ainda os conceitos defendidos pelo modelo arquitectural presente em SOA e a metodologia de desenvolvimento subjacente a Software as a Service. No capítulo 3 são apresentadas algumas ferramentas de apoio à constituição de *mashups*, assim como trabalhos relacionados com formas de disponibilização de serviços com o objectivo de facilitar aos clientes a procura desses serviços.

Esta dissertação propõe um modelo arquitectural para a construção de sistemas de mediação de serviços. Os capítulos 4, 5 e 6 são dedicados a esse tópico. No capítulo 4 descreve-se a fase de concepção do modelo de mediação. São abordados aspectos relevantes para a constituição do modelo, sendo explicadas as decisões tomadas na sua elaboração. No capítulo 5 é apresentado o modelo arquitectural. São descritos os módulos presentes

na sua constituição assim como as metodologias e processos de execução a si inerentes. No capítulo 6 apresenta-se um exemplo de implementação do modelo proposto. Como tal, são exemplificados os processos de construção dos diversos módulos apresentados no capítulo anterior. Ainda neste capítulo e de forma a testar a implementação concebida, é ainda apresentada uma aplicação exemplo, assente nos princípios de *mashup*, que utiliza o sistema de mediação como única fonte de obtenção de conteúdos.

O capítulo 7 conclui a dissertação, sumariando o trabalho realizado e dele retirando algumas ilações finais. São ainda apresentadas direcções que permitem dar continuidade ao trabalho realizado.

Capítulo 2

Contextualização

2.1 Web Services

Inicialmente, Web Services foram definidos pela *World Wide Web Consortium* (W3C) como aplicações modulares, auto-contidas e auto-descritas desenvolvidas com base em standards abertos [W3Cf]. No entanto, nos últimos anos a própria W3C mudou essa definição, apesar de ainda não a ter tornado formal, ao descrever Web Services categoricamente como um sistema de software.

Na verdade, a tecnologia de Web Services tem sofrido um crescimento bastante acentuado nos últimos anos. A explicação para tal fenómeno é facilmente perceptível devido ao posicionamento da própria tecnologia como a mais adequada para sistemas distribuídos [Gur04].

De facto, a indústria de sistemas de informação tem-se debatido, ao longo do tempo, com problemas derivados de integrabilidade em sistemas distribuídos [Gur04]. Como tal, a solução mais óbvia seria de facto fazer com que todas as aplicações implementassem a mesma solução de integração. Assim, e apesar de terem surgido nos anos 90 standards como CORBA [Grob] ou COM [Cora], estes não conseguiram providenciar totalmente uma framework de integração com que todos concordassem. Actualmente, os Web Services tornaram-se de facto nessa mesma solução.

O nascimento dos Web Services como paradigma de desenvolvimento de aplicações, e mesmo arquitecturas, surgiu logo após o crescimento das aplicações Web. Aplicações Web podem ser definidas como aplicações acessíveis através de um browser web, ou similar, que permitem a qualquer utilizador, em qualquer parte do mundo, o acesso às suas funcionalidades. De facto, aquando do seu aparecimento, as diferenças em relação a aplicações mais antigas baseadas no modelo cliente-servidor, originaram um forte crescimento, passando de centenas de utilizadores em modelos cliente-servidor para milhões de utilizadores. Por

conseguinte, a Web tornou-se no meio principal de acesso para aplicações, muito devido à necessidade por parte dos utilizadores de apenas especificarem um URL num browser [CW03].

No entanto, a dificuldade no desenvolvimento de aplicações também aumentou. Um cliente de uma aplicação Web, não possui qualquer conhecimento acerca dos requisitos das comunicações ou sobre os próprios sistemas. Assim, tecnologias standard como `HTTP` [W3Cb] ou `HTML` [W3Ca] foram utilizadas para preencher este vazio entre as aplicações e seus clientes, enquanto servidores aplicacionais ou mesmo middleware específico surgiram como resposta à complexidade no desenvolvimento das aplicações. Os Web Services de alguma forma, estendem este modelo. De facto, enquanto que as aplicações Web limitam-se a permitir que um browser aceda às suas funcionalidades através de uma interface específica à aplicação, os Web Services por sua vez, permitem que qualquer aplicação cliente aceda e utilize as suas funcionalidades directamente como se fizessem parte do seu sistema [CW03].

A necessidade da utilização de Web Services provém essencialmente da sua capacidade para alterar de forma dramática, mas positiva, aspectos relativos às aplicações, inclusive em processos de oferta ou de procura. Na verdade, a utilização de Web Services reduz significativamente os custos de desenvolvimento de aplicações sobretudo através da sua capacidade de inclusão de funcionalidades de software de terceiros através da Web. Deste modo, permite-se que conteúdos valiosos presentes em aplicações já existentes sejam facilmente isolados e reutilizados. Assim, e devido a serem unicamente centrados na Web, programadores que procurem as melhores implementações de software, podem facilmente aceder a essas mesmas implementações sem terem que se preocupar com possíveis entraves a nível geográfico, ou mesmo político. Por conseguinte, através da rápida adopção dos Web Services, aplicações situadas em diversas localizações da Web podem ser directamente integradas e interligadas como se fizessem parte de um sistema singular [New02], como é ilustrado na Figura 2.1.

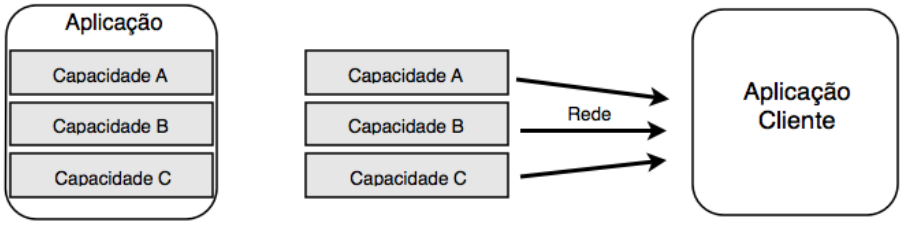

Figura 2.1: Inclusão de algumas capacidades numa aplicação monolítica (esquerda) e numa aplicação cliente, via Web Services (direita)

Desta forma, erros comuns provenientes da complexidade de grandes aplicações mono-

líticas são reduzidos ou mesmo eliminados, originando assim também, reduções no tempo de desenvolvimento do software e na rapidez com que este atinge o mercado [CW03]. Para além da redução dos erros, a manutenção das próprias aplicações é simplificada, através de uma segmentação efectuada à aplicação, resultando em duas partes, a aplicação cliente e seus respectivos Web Services.

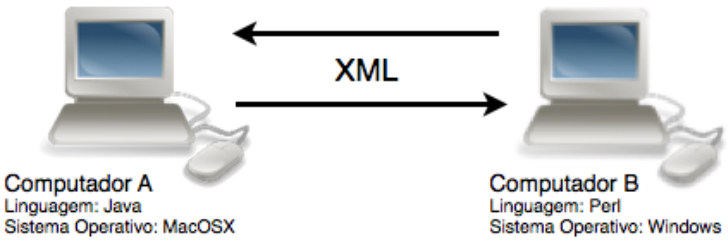

Figura 2.2: Exemplo Básico de um Web Service

Como é visível na Figura 2.2, a implementação de um Web Service é tradicionalmente muito simples e com baixa complexidade. Características próprias dos Web Services, entre as quais, independência na plataforma utilizada, assim como independência nos sistemas operativos utilizados, originam uma acrescida rapidez na sua implementação. Se a isto aliarmos a modernização de ferramentas capazes de gerar todo o código necessário para a implementação de um Web Service, esta simplicidade pode ser exponenciada.

Outra aplicabilidade dos Web Services provém dos designados sistemas *legacy*. Estes sistemas, centram o seu modelo de negócio num processamento que se inicia e termina nessa mesma aplicação, ou seja, não possuem qualquer tipo de interoperabilidade com o exterior. Assim e apesar de estas aplicações cumprirem os objectivos a que se propõem, não se encontram endereçadas para as necessidades de hoje [Gur04]. Deste modo, a modernização destes sistemas através da utilização de Web Services, para além das novas oportunidades de negócio que possam representar, proporciona aos seus proprietários a recriação de lógicas de negócio com créditos firmados em novas aplicações prontas para os desafios do mercado actual. Como tal, estes sistemas tornam-se automaticamente como candidatos para se tornarem fornecedores de Web Services.

2.1.1 Metodologia de Implementação de um Web Service

A constituição de um Web Service condiciona habitualmente a existência de três entidades principais, como se ilustra na Figura 2.3, através das quais todo o Web Service será gerido. Estas entidades são [Cer02]:

- Fornecedor do Serviço *(Service Provider)* - entidade responsável por implementar o

serviço e torná-lo disponível através da Internet.

- Cliente do Serviço *(Service Requester)* - entidade que invoca o Web Service. O cliente utiliza o serviço através de uma ligação pela Web, normalmente através de um pedido `XML`.
- Registo de Serviços *(Service Registry)* - entidade que providencia um local central onde os serviços são publicados. No fundo, trata-se de um directório centralizado de serviços. Devido a novas implementações de Web Services, a utilização de um directório de serviços centralizado tem sido de alguma forma descontinuada.

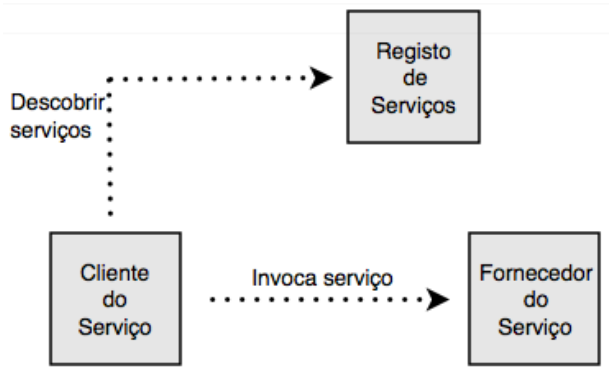

Figura 2.3: Partes Fundamentais de uma arquitectura de Web Services

Uma Web Service Protocol Stack é uma pilha protocolar usada para ajudar na construção de um Web Service, tendo em consideração as entidades anteriores. Tipicamente define quatro protocolos:

- *Serviço de Transporte*, responsável pelo transporte das mensagens entre as aplicações. Normalmente efectuado através de `HTTP`, apesar da ser possível a utilização de `FTP` e de `SMTP`.
- *Protocolo de Mensagem*, responsável pela codificação da mensagem de forma a que possa ser percebida pelos dois lados da ligação. Dependendo da arquitectura a utilizar pode variar em `XML`(`SOAP`), `XML` puro ou mesmo outros formatos como `MIME` ou `JSON` (apenas em serviços RESTful).
- *Descrição do Serviço*, referente à descrição da interface pública com um web service, i.e., de como pode o cliente ligar-se com o serviço. Esta descrição é normalmente efectuada através do protocolo `WSDL` ou textualmente.
- *Descoberta de Serviço*, refere-se ao registo onde os serviços podem publicar a sua localização e descrição. Um exemplo de registo central é o `UDDI`.

Por forma a implementar um Web Service, existe nos nossos dias uma clara divisão entre as duas principais arquitecturas disponíveis: os designados Big Web Services, ou seja, a pilha SOAP/UDDI/WSDL e os serviços RESTful.

2.1.2 "Big" Web Services - SOAP/WSDL/UDDI

A pilha WS-* surgiu em 1999 por iniciativa da Microsoft, através dos seus constituintes mais conhecidos: *Simple Object Access Protocol* (SOAP) [W3Cc] e *Web Services Description Language* (WSDL) [W3Cd], com o intuito de se tornar uma alternativa a tecnologias anteriores como CORBA ou DCOM. Correntemente na versão 1.2, o SOAP tornou-se no standard em trocas de mensagens baseadas em XML. No entanto, trata-se de uma tecnologia complexa, muito devido à sua extensibilidade que surge sob a forma de WS-* standards. Exemplos destes são WS-Addressing, WS-Security ou WS-ReliableMessaging entre muitos outros. Estes standards, apesar de não serem obrigatórios na implementação básica de SOAP, são muitas vezes necessários para preencher lacunas ou adicionar funcionalidades ao protocolo original. Por forma a utilizar este tipo de Web Services, um modelo de utilização utilizando SOAP, WSDL e UDDI semelhante à Figura 2.4 será necessário.

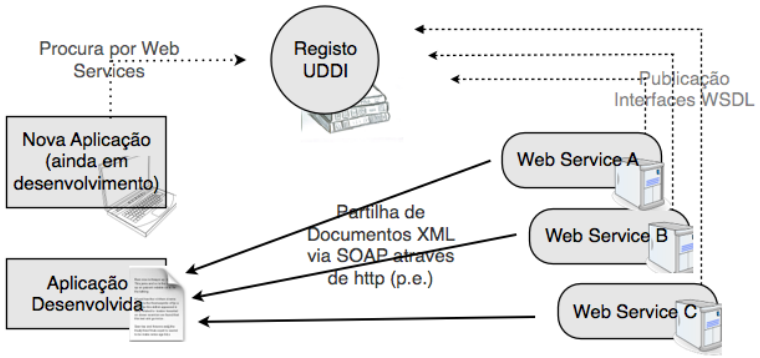

Figura 2.4: Modelo Genérico de Utilização de Web Services com XML, SOAP, WSDL e UDDI

O SOAP representa uma envelope para as mensagens trocadas entre os intervenientes do Web Service. Assim, por forma a realizar um pedido, o cliente terá que compor uma mensagem SOAP e enviá-la ao servidor. Por sua vez, o servidor irá interpretar a mensagem e posteriormente responder segundo o mesmo formato. A estrutura básica de uma mensagem SOAP é constituída em XML por um elemento superior *envelope*, que por sua vez contém um *header* e um *body*. O *header* é opcional, podendo conter informação para routing ou aspectos relacionados com qualidade de serviço (QoS), tais como encriptação ou fiabilidade da transferência. Por sua vez, o *body* da mensagem contém apenas a informação que se pretende transmitir.

Por forma a enviar a mensagem, o cliente terá que saber como comunicar com o servidor, ou seja, que métodos e respectivos argumentos terá que invocar de forma a efectuar o pedido. Para tal o cliente utilizará a interface do serviço descrita no documento `WSDL`. Este documento tem como propósito dispor informação relativa à funcionalidade do Web Service assim como descrever a interface do serviço. Através de uma leitura do documento `WSDL`[1], o cliente passará a saber como poderá invocar o Web Service e simultaneamente as suas características sintácticas. A estrutura de um documento `WSDL` é constituída por duas secções, uma concreta e uma abstracta. A parte abstracta contém a estrutura das mensagens, ou seja, descreve o tipo de dados que serão enviados durante as trocas de mensagens. Para além das mensagens, a parte abstracta do `WSDL` compreende ainda a descrição de todas as operações passíveis de serem invocadas. Cada tag *operation* presente no documento representa um método acessível pelo serviço, sendo que, por forma a facilitar a integrabilidade com a parte concreta do documento, estas tags são agrupadas de modo a representarem um conjunto de operações (recorrendo à tag *portType*).

A parte concreta do documento é composta primeiramente por uma secção designada por *binding*, onde é especificado o tipo de interacção e protocolo de comunicações a utilizar para cada elemento *portType* descrevendo deste modo a forma como cada uma das operações poderá ser utilizada. Por fim, a secção *service* agrupará todas as descrições correspondentes a cada ponto de acesso implementado, através do qual cada *portType* possa ser acedido pelos clientes utilizando por exemplo o `URI` descrito.

Por forma a utilizar o serviço, o cliente terá que desenvolver uma espécie de *stub*[2], com base no `WSDL`, que lhe permita comunicar com o servidor. De salientar que actualmente este *stub* pode ser automaticamente gerado devido a características standards do `WSDL`, facilitando assim o grau de integração com Web Services, assim como abrindo caminho para a criação de sistemas dinâmicos que utilizem Web Services.

De facto, para além da capacidade do `WSDL` poder ser totalmente processado pelo sistema, a sua utilização atribui ainda um elevado nível de abstracção a nível do protocolo de comunicação ou mesmo a detalhes de serialização, sendo que adicionalmente a própria plataforma de serviço utilizada, quer a nível do sistema operativo, quer a nível da linguagem programacional utilizada, torna-se completamente transparente para o cliente [PZL08].

Apesar da utilidade do formato `WSDL`, foi entendido na sua concepção que seria necessário um local onde este pudesse ser publicado de modo a possíveis clientes o poderem

[1] escrito em XML

[2] *stub* - algoritmo que estabelece uma ligação entre a aplicação cliente e o mecanismo de comunicação do Web Service.

encontrar.

O UDDI foi proposto como método de registo sendo definido como um mecanismo universal, aberto e independente de plataforma capaz de fundamentalmente providenciar uma forma standard, uniforme e rapidamente acessível para aplicações via APIs ou interfaces gráficas (GUIs), através da qual empresas e organizações que disponibilizem Web Services possam descrever e publicar os serviços. Adicionalmente, o UDDI foi proposto como algo capaz de providenciar informação tecnológica de como pode um serviço ser localizado, acedido ou mesmo utilizado. Assim, programadores em busca de serviços que correspondam aos seus objectivos, podem encontrar no UDDI a solução que pretendem [RR07].

De facto, uma mais-valia de utilização do UDDI centrar-se-ia nas aplicações, que, se correctamente implementadas, poderiam utilizar o UDDI como forma de encontrar um serviço que pretendam e incorporá-lo imediatamente (através da ligação ao WSDL), sendo tal acção transparente para o utilizador.

No entanto, apesar das suas inúmeras vantagens, o UDDI nunca se conseguiu afirmar, sendo que alguns factores apontados para tal se centram no baixo número de serviços SOAP públicos ou na capacidade do WSDL de promover uma única interface para cada Web Service.

A especificação do UDDI é definida através de um *schema* XML contendo quatro camadas hierárquicas possuindo meta-informação relativa às descrições dos serviços. Cada registo UDDI é composto na sua totalidade por quatro tipos: *businessEntity*, *businessService*, *binding-Template* e *tModel* [RR07]

2.1.3 RESTful

O estilo arquitectural designado por *Representational State Transfer* (REST) foi introduzido por *Roy Fielding* na sua dissertação de doutoramento [Fie00] no ano 2000. Na sua essência, REST propõe-se a simplificar o conceito de Web Service tendo por base uma implementação com menor grau de complexidade e fundamentalmente capaz de fornecer um meio mais fácil de implementação e utilização de Web Services.

No centro da sua utilização encontram-se recursos[3], que não possuem uma forma concreta, podendo ser aquilo que o programador pretender. De facto, um recurso pode assumir formas tão distintas como um ficheiro pdf, um número, um endereço de um site ou o próprio site ou até mesmo uma imagem. Através deste mecanismo, o REST obtém uma maior abrangência relativamente aos formatos de transferência, incrementando

[3]resources

exponencialmente as possíveis utilizações face aos Web Services tradicionais [RR07].

A grande capacidade inerente aos recursos provém do seu identificador, o *Uniform Resource Identifier* (URI). De facto, é o URI que define um recurso atribuindo-lhe o nome, assim como o seu endereço. Assim, o URI torna-se no meio através do qual o serviço expõe a informação desejada na Web, assim como a forma através da qual um cliente pode seleccionar um determinado conteúdo. Por forma a constituir um recurso, a cada conteúdo será atribuído um URI exclusivo, sendo que apesar da exclusividade relativa ao URI face a um recurso, este poderá possuir vários identificadores.

A utilização de URIs atribui inúmeras diferenças à arquitectura RESTful face ao modelo SOAP. Através de URIs, REST prova que é possível descobrir conteúdos em Web Services sem a necessidade de um registo centralizado (como o UDDI) dando origem a que actualmente os equivalentes mais próximas para REST sejam mesmo simples motores de pesquisa, como o Google ou o Yahoo [PZL08]. Adicionalmente, possibilita aos futuros clientes a capacidade de acederem aos seus serviços a partir de um browser, sem a necessidade de existirem clientes dedicados. De facto, a sua emergência acabou por ser também uma reacção à dificuldade de implementação do protocolo SOAP, desde logo constituindo-se como uma alternativa mais simples.

Na verdade, a escalabilidade e boa formação da Web actual são outros dos principais conceitos pelo qual se rege a arquitectura REST, originando a que a sua implementação não necessite de uma infra-estrutura própria e dedicada, mas apenas um servidor (e cliente) HTTP.

Simultaneamente, a utilização do HTTP como protocolo de transporte é também um dos principais factores para tal. Nos nossos dias, a Web predomina sobre todos os restantes protocolos muito devido à sua capacidade de etiquetar qualquer item disponível. Através de URIs e HTTP, o REST tira partido desse domínio.

Assim, o modelo de operação de serviços REST assenta essencialmente sobre o protocolo HTTP utilizando-o para a transferência de informação. Mas a informação que se pretende transferir não são os recursos. De facto, os recursos são apenas algo abstracto relativo à informação, como por exemplo uma imagem de Paris, mas não se referem a um série de bytes ou a um tipo de ficheiro. Tal designa-se por representação do recurso.

Um recurso pode ser definido como uma fonte de representações, e uma representação como alguns dados sobre o estado actual do recurso. Apesar disto, a maioria dos recursos retratam dados por si mesmos, originando que a representação do recurso sejam os próprios dados [RR07].

A manipulação destas representações é efectuada com base nas operações suportadas pelo HTTP através da seguinte correspondência:

- *Put* - cria uma representação para um novo recurso;
- *Get* - procura e entrega o estado actual de um recurso segundo uma representação;
- *Post* - modifica o estado actual de um recurso;
- *Delete* - apaga o recurso seleccionado;

Assim, as representações são utilizadas pelos serviços, sendo que devido à sua estruturação auto-descritiva através de meta-informação associada à representação, permite que os recursos possam ser dissociados da sua representação e por conseguinte possam ser acedidos segundo diversos formatos pelo cliente (ex. JPEG, XML, texto, etc.)[PZL08].

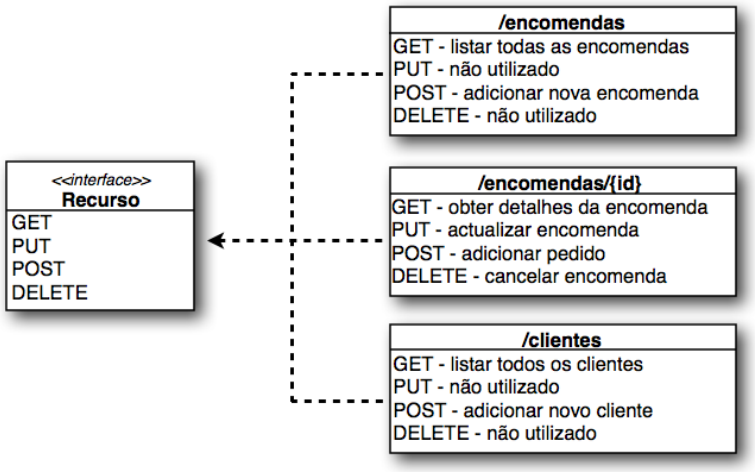

Figura 2.5: Utilização de Recursos na arquitectura RESTful

De facto, recorrendo à Figura 2.5, é visível que numa analogia a um diagrama de classes em UML [Groa], em REST um recurso pode ser abstraído como sendo simplesmente uma interface com quatro métodos que será implementada por todos os recursos envolvidos no Web Service. No entanto, a implementação dos métodos varia de recurso para recurso, sendo que é possível que determinados recursos não implementem alguns dos métodos.

Assim cada recurso implementará a interface principal, e recorrendo às práticas de boa implementação de REST, utilizará um endereço elaborado numa perspectiva lógica por forma a simplificar a leitura dos mesmo por parte dos clientes.

Na verdade, a implementação de endereços (`URIs`) em `REST` é efectuada numa perspectiva lógica, ou seja, de forma a que a leitura do URI dê a perceber o que pode ser obtido por esse recurso. Por exemplo, utilizando os recursos presentes na Figura 2.5, *"/clientes"* oferece a percepção que será efectuado um pedido direccionado para todos os clientes da aplicação. Se adicionarmos "id" ao endereço já existente, da seguinte forma

"/clientes/id", então será claro para o utilizador que se está a manipular um cliente específico. Por esta razão, os utilizadores de `REST` são encorajados a utilizar endereços que de certa forma, ofereçam alguma informação adicional sobre o recurso, facilitando a tarefa a futuros clientes.

Devido ao fácil endereçamento inerente à utilização de `URI`s, o `REST` não possui um registo centralizado, originando que a descrição de um serviço RESTful seja normalmente efectuada textualmente e a sua obtenção através de websites como o *Programmable Web*. No entanto, a partir da versão 2.0, o `WSDL` já suporta a descrição de interfaces `REST` para além de `SOAP` mas no entanto continua a não existir nenhum modelo centralizado para publicação das mesmas à semelhança do `UDDI` em serviços `SOAP`.

2.1.4 Análise Comparativa entre SOAP e REST

Na Figura 2.6 são apresentadas algumas comparações entre os dois tipos de implementações sob a forma de tecnologias disponíveis actualmente. Através desta tabela é possível efectuar uma análise relativamente aos seus modos de operação e principais áreas de focalização.

Architectural Decision and AAs	REST	WS-*
Transport Protocol	1 AA	≥7 AAs
HTTP	✓	✓
waka [13]	✓	
TCP		✓
SMTP		✓
JMS		✓
MQ		✓
BEEP		✓
IIOP		✓
Payload Format	≥6 AAs	1 AA
XML (SOAP)	✓	✓
XML (POX)	✓	
XML (RSS)	✓	
JSON [10]	✓	
YAML	✓	
MIME	✓	
Service Identification	1 AA	2 AA
URI	✓	✓
WS-Addressing		✓
Service Description	3 AAs	2 AAs
Textual Documentation	✓	
XML Schema	✓	✓
WSDL	✓	✓
WADL	✓	

Architectural Decision and AAs	REST	WS-*
Reliability	1 AA	4 AAs
HTTPR	(✓)	(✓)
WS-Reliability		✓
WS-ReliableMessaging		✓
Native		✓
Do-it-yourself	✓	✓
Security	1 AA	2 AAs
HTTPS	✓	✓
WS-Security		✓
Transactions	1 AA	3 AAs
WS-AT, WS-BA		✓
WS-CAF		✓
Do-it-yourself	✓	✓
Service Composition	2 AAs	2 AAs
WS-BPEL		✓
Mashups	✓	
Do-it-yourself	✓	✓
Service Discovery	1 AAs	2 AAs
UDDI		✓
Do-it-yourself	✓	✓
Implementation Technology	many	many
...	✓	✓

Figura 2.6: Comparação de Tecnologias REST e WS-* [PZL08]

Relativamente aos protocolos de transporte suportados é desde logo visível uma antítese entre as duas implementações. Enquanto serviços RESTful apenas suportam `HTTP` (waka está ainda em desenvolvimento), WS-*, é capaz de operar em alguns mais. Enquanto isto é sem dúvida uma vantagem para o lado do WS-* devido a uma menor tolerância a falhas através do suporte a vários protocolos, a implementação de SOAP em

clientes diferentes requer obrigatoriamente trabalho adicional. No entanto isto pode ser analisado inversamente dado que, em serviços RESTful qualquer cliente HTTP comunica com outro sem necessidade de configurações adicionais simplificando assim a comunicação e a sua implementação.

Já relativamente aos formatos da mensagem, acontece o inverso, visto a stack WS-* apenas utilizar `XML`(`SOAP`), garantindo uma uniformidade relativamente ao tipo de mensagem utilizada. Contrariamente, `REST` possui uma maior abrangência, e enquanto tal pode complicar a implementação devido a uma possível incompatibilidade entre formatos esperados pelo cliente e aquele, que de facto recebe, podendo também ser um aspecto positivo se consideramos a possibilidade de utilização de formatos `MIME` ou mesmo `JSON` [JSO].

A fiabilidade das duas implementações é semelhante nas suas configurações mais simples. No entanto, o `SOAP` através da sua extensibilidade, possibilita a adição de standards WS por forma a obter garantia de uma maior tolerância a falhas. Para além da fiabilidade, o mesmo sucede com aspectos relacionados com a integridade da informação tais como a segurança ou mecanismos de transacção onde mais uma vez, a existência de standards adicionais para `SOAP` atribui-lhe maior garantia. No entanto para o programador, a inclusão destes standards complica substancialmente a implementação do serviço, além de que a sua integração, devido à sua complexidade só poderá ser efectuada com recurso a ferramentas.

Outro aspecto relevante a salientar é a inexistência de um protocolo para descoberta de serviços em `REST`. Deste modo, torna-se necessário que a procura pelos mesmos ocorra manualmente. No entanto, apesar da existência do `UDDI` [OASb] para `SOAP`, o seu grau de utilização é relativamente baixo, muito devido à falta de inclusão de características não-funcionais nas suas descrições [Ran03].

Por último, a existência de um formato standard para descrição de propriedades sintácticas (`WSDL`) é outro ponto importante em favor de `SOAP`, sendo que para utilizar o Web Service o cliente apenas tem que obter o documento `WSDL` do serviço pretendido, e através das capacidades de *binding* automáticas, incorporá-lo no seu sistema. Já RESTful através da criação do formato `WSDL` 2.0 começa a caminhar nessa direcção, sendo que em seu benefício, uma nota a assinalar centra-se com o crescimento registado na construção de mashups devido à sua utilização.

2.2 Mashup

O surgimento da Web 2.0 e consequente crescimento das tecnologias associadas, deram origem a novas e inovadoras formas de colaboração e desenvolvimento para desafios já existentes baseando-se em conceitos como comunidade, usabilidade, organização e apresentação como alguns dos seus pilares [RHC08]. De facto, a Web tem originado uma mudança de paradigma na forma como o software é desenvolvido nomeadamente através da proliferação de `APIs` públicas e sua crescente disponibilização [MRT07] que, através da sua combinação, torna possível a criação de soluções para dar resposta a problemas específicos ou mesmo esporádicos. Uma destas soluções designa-se por *Mashup*.

Um mashup pode ser definido simplesmente como uma aplicação web capaz de combinar informação proveniente de diversos locais da Web, normalmente através de Web Services, e de providenciar conteúdos únicos, sendo estes disponibilizados através de uma interface integradora [Zan08]. Assim, toda a estrutura inerente a um mashup assenta sobre a composição de conteúdos que quando "sobrepostos" transmitem ao utilizador informação mais rica sob uma interface diferente.

Deste modo, mashups tornam-se mais do que uma simples combinação entre dois ou mais websites, mas um método através do qual os programadores são capazes de utilizar sistemas já existentes por forma a criar novas e inovadoras maneiras de originar novos conteúdos.

Exemplos de mashups incluem inclusive widgets utilizados no ambiente de trabalho, que diferem dos mashups tradicionais pelo facto de disporem de uma interface dedicada e não necessitarem de um browser, ou ferramentas como o *ubiquity*[Moz] que fornecem ao utilizador um meio através do qual pode realizar queries semânticas no próprio browser e obter os resultados dinamicamente.

Figura 2.7: Exemplo de um mashup

Alguns mashups típicos incluem por exemplo a junção de informação proveniente de diversas lojas numa única lista por forma a ajudar os consumidores a encontrar o melhor preço de um determinado produto. Outros, como o visível na Figura 2.7, adicionam informação relativa às temperaturas num mapa geográfico.

Um bom exemplo do valor acrescido dos mashups é ilustrado por este último caso, onde a informação relativa às temperaturas é disponibilizada de acordo com uma localização geográfica existente. Assim, e aliado ao conceito de mashup, a informação recolhida passará a ser disposta sobre um mapa permitindo assim ao utilizador facilmente associar a temperatura recebida a uma região pretendida. Adicionalmente, com a inclusão de diversas entradas sobre o mapa, o utilizador poderá estabelecer comparações numa determinada região simplesmente por navegar no mapa.

Este tipo de mashups, ou seja, compostos através da síntese de mapas enquadra-se numa das duas principais áreas de desenvolvimento de mashups designada por Sistemas de Informação. De facto, alguns estudos relativamente aos tipos de mashups com maior sucesso indicam claramente um domínio por parte dos relacionados com serviço de mapas (aproximadamente 80%) e com fotografia (com 40%), sendo que alguns incorporam as duas componentes [Zan08].

Para além da síntese de mapas, mashups com propósitos de integração de informação empresarial ou descoberta de web services enquadram-se neste área. O seu principal objectivo centra-se na identificação da informação mais relevante e sua posterior extracção por forma a constituir os requisitos dos sistemas de informação [Zan08].

A outra principal área de desenvolvimento designada *End-User Programming* (`EUP`), dispõe-se fundamentalmente a permitir que pessoas que possam escrever código mas não sejam programadores profissionais, sejam capazes de utilizar técnicas de programação como meio de atingirem os seus objectivos. Assim, e devido ao facto de os mashups comporem essencialmente a manipulação de informação, investigação em `EUP` tem surgido sobretudo através da criação de ferramentas de ajuda à construção dos mesmos (mashup makers) [CLND09].

2.2.1 Construção de um mashup

Independentemente da utilização ou não de ferramentas de apoio ao desenvolvimento, o processo de criação de um mashup é baseado em alguns passos primários consoante é ilustrado na Figura 2.8.

Por forma a construir um mashup, inicialmente o programador tem que percorrer um processo de selecção e agregação de informação proveniente de diversas fontes. De facto,

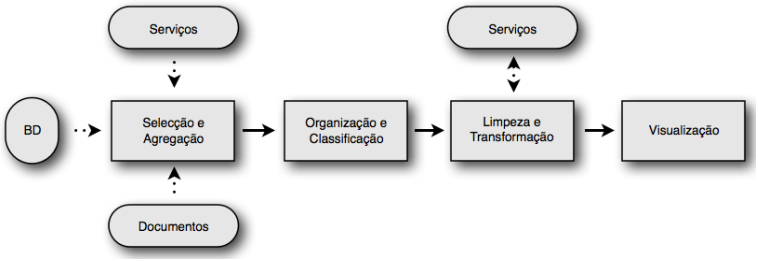

Figura 2.8: Processo de produção de um mashup [MMD06]

as fontes de informação de um mashup podem ser muito variadas, nomeadamente a nível de formato (`HTML`, `PDF`, entre outros), estrutura ou até mesmo localização (ficheiros locais ou a web) [MMD06].

É improvável que um mashup utilize na totalidade os conteúdos de uma só fonte, ou seja, normalmente apenas se utilizam fragmentos de uma mesma fonte. Deste modo, torna-se importante que o programador tenha em atenção as fontes que selecciona, dado que dependendo da quantidade de informação que possa ou não necessitar, terá obrigatoriamente que processar essa informação. Recorrendo ao exemplo apresentado anteriormente, o programador poderia neste passo procurar determinados serviços que fossem capazes de lhe fornecer temperaturas com base em determinadas localizações geográficas.

De seguida, o programador centra-se-á na organização e classificação da informação recolhida. Recorrendo de novo ao exemplo, o programador pode organizar a informação referente às temperaturas e agrupá-las de acordo com o continente correspondente ou uma área por si definida sendo que o principal objectivo será a redução na quantidade de informação obtida.

No próximo passo, o programador ocupar-se-á da transformação da informação recolhida sendo esta processada por forma a poder ser utilizada. Para tal, a informação atravessa um processo semelhante a um parsing com vista à obtenção de uma estrutura utilizável, sendo que para tal, o programador pode recorrer a um serviço externo para obter essa mesma estrutura. Assim, por exemplo, o programador pode compor a informação obtida relativamente às temperaturas num formato standard (p.e. `XML`) de maneira a que possa ser sobreposta num mapa.

Por fim, no passo de visualização ou de integração, o programador implementa o código necessário de forma a construir a interface. Assim, e terminando o exemplo, o programador sobrepõe o resultado composto no passo anterior sobre um mapa disponibilizado através uma `API` pública (p.e. Google Maps) concluindo assim a construção do mashup.

Normalmente, a integração de informação proveniente de uma empresa e de outras aplicações numa nova aplicação coerente e com valor próprio requer uma alta capacidade

de programação, para além de um conhecimento elevado sobre as fontes de informação e lógicas de negócios envolvidas. Apesar de novas tecnologias como `AJAX` e RESTful web services, ou mesmo formatos como `RSS` feeds, simplificarem a construção de mashups, estas ainda requerem que o programador as domine, o que por si só limita o possível espectro de programadores.

Como tal, e por forma a simplificar a criação de mashups, tem surgido um leque de ferramentas de desenvolvimento específicas à elaboração de mashups. Conhecidos como *Mashup Makers*, são exemplos o Microsoft Popfly[Mica], Yahoo Pipes [Yah] ou o Google Mashup Editor [Goob].

Estas ferramentas proporcionam uma maior facilidade de construção de mashups a utilizadores menos experientes, possuindo, no entanto, cada uma o seu público alvo. Isto acontece simplesmente porque algumas destas ferramentas são orientadas a programadores enquanto outras possuem como alvo o utilizador comum.

Na globalidade, estas ferramentas pretendem maioritariamente libertar o utilizador da componente de comunicações com os Web-Services através de uma camada de abstracção integradora ou fornecer elementos base para a construção da interface gráfica.

2.2.2 Propriedades

Apesar da simplificação providenciada por estas aplicações, normalmente estas apenas se relacionam com o processo de construção do mashups. Como tal, aspectos como a confiabilidade das `APIs` por exemplo, podem-se tornar críticos para os mashups fazendo com que independentemente da utilização das ferramentas ou não, o programador tenha ainda obrigatoriamente que analisar certos parâmetros relacionados com o ambiente de execução do mashup. Para tal, são definidas quatro propriedades: *deployment*, *local de execução da aplicação*, *requisitos da aplicação* e *escalabilidade* do ambiente de execução [YBCD08]:

- o *deployment* dependerá apenas na forma como o mashup for implementado, podendo tal acontecer de uma forma isolada, num Web-server administrado pelo programador do mashup ou mesmo através de um Web-server privado.

- dependendo das tecnologias utilizadas para o desenvolvimento do mashup, a *execução da aplicação* poderá ocorrer no lado do servidor (p.e. `PHP`), no lado do cliente (p.e. JavaScript) ou nos dois. Neste último caso, normalmente a composição é efectuada no lado do servidor sendo que do lado do cliente apenas são apresentados os resultados da aplicação.

- relativamente aos *requisitos do sistema*, devido à componente de utilização do browser, por vezes torna-se necessário que o browser do utilizador possua determinados plugins ou mesmo extensões ao mesmo. Assim, por vezes torna-se necessário utilizar um determinado browser em favor de outro, dependendo do ambiente de desenvolvimento que o programador utilizou.

- por fim, a *escalabilidade* do sistema. Geralmente, problemas de escalabilidade não surgem se a execução da aplicação ocorrer no lado do cliente. No entanto, se tal ocorrer no lado do servidor, uma análise relativamente ao número de utilizadores do mashup assim como o número de composições ou fontes de informação, terá que ser efectuada por forma a medir a escalabilidade do mesmo.

2.3 SOA — *Service Oriented Architecture*

Nos últimos anos, a arquitectura *Service Oriented Architecture* (SOA) evoluiu, deixando de ser apenas um conceito promovido para se estabelecer como um dos mais importantes estilos arquitecturais [IHJ+07]. Trata-se de uma atitude na concepção de aplicações e componentes de software. Pode ser descrita como um modelo arquitectural distribuído que tem por base o "encapsulamento" de aplicações em serviços que por sua vez comunicam entre si através de um sistema de comunicação inerente à própria arquitectura.

As soluções SOA possuem como base de concepção os serviços de onde surgem dois protagonistas, o requisitor do serviço (cliente) que procura o serviço, e o fornecedor do serviço que por sua vez disponibiliza e em última instância fornece o serviço [PH07].

Um serviço, por sua vez, não é mais que um tipo comportamental providenciado por um componente para possibilitar a sua utilização por um outro componente qualquer com base na sua interface contratual. Assim, é necessário que um serviço em SOA, possua uma interface acessível por rede para assim ser possível que um cliente ou serviço possa invocá-lo [Steb]. Deste modo, apesar de os serviços SOA serem visíveis para o cliente, os respectivos componentes são transparentes.

Apesar de em SOA ser essencialmente necessário que existam o cliente e o fornecedor como serviços, tal poderia levar a problemas de selecção devido à escolha que um cliente tem que efectuar por entre um vasto número de aplicações disponíveis. Assim, e no sentido de obter a melhor escolha, um novo papel surge sob a forma de um intermediário entre o cliente e o fornecedor do serviço, o mediador.

Tal como ilustrado na Figura 2.9, o papel do mediador é o de um intermediário (um exemplo seria um registo UDDI), onde os fornecedores de um serviço publicar as suas

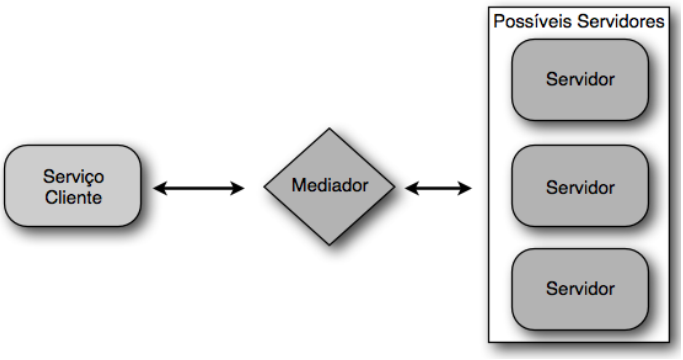

Figura 2.9: Mediador na estrutura em SOA

interfaces referentes ao que oferecem e onde os clientes encontram informações sobre os serviços disponíveis. Para além deste papel, o mediador pode ainda ter um papel de selecção entre os serviços com base em aspectos relacionados com confiabilidade, qualidade de serviço, contratos de serviço, entre outros, que estariam assim inerentes ao próprio mediador [PH07].

A arquitectura `SOA` pode ser dividida em diferentes camadas de abstracção de forma a poder ser discutida de um ponto de vista conceptual. As quatro camadas básicas, ilustradas na Figura 2.10, incluem [TV08]:

- A *camada aplicacional*, que inclui por exemplo, sistemas mais antigos (legacy), *Customer Relationship Management* (`CRM`), *Enterprise Resource Planning* (`ERP`) ou bases de dados adicionais.

- A *camada de serviços* onde os serviços são disponibilizados com base na camada aplicacional e normalmente através de interfaces descritas em `WSDL`.

- A *camada de processamento* onde os serviços são orquestrados, ou seja, são compostos de forma a comporem um processo sob a forma de processos, por exemplo *Business Process Execution Language* (`BPEL`) [OASa].

- Finalmente, a *camada de apresentação* disponibiliza aos utilizadores as funcionalidades de `SOA` através de aplicações desktop ou aplicações Web (portais).

`SOA` como arquitectura propõe-se a vários objectivos. Um deles é agregação de lógicas de negócio complexas assentes sobre interfaces de serviço standard, ou seja, a fácil obtenção de novas aplicações com base em serviços através de simples integração da interface correspondente. Deste modo, a implementação da camada de negócio é escondida não tendo assim importância para o desenvolvimento da aplicação [Nes08].

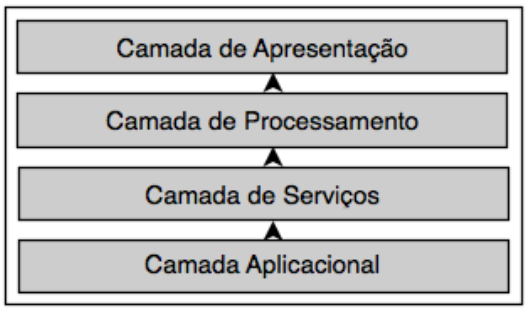

Figura 2.10: Camadas de Abstracção em SOA

Outro objectivo centra-se com a interoperabilidade expressa através da capacidade de qualquer componente permitir que eventualmente possa ser invocado por um qualquer potencial cliente do mesmo [Steb]. Tal é obtido em SOA através da obrigatoriedade de cada componente providenciar uma interface que possa ser invocada através de um formato por mensagens assente sobre um protocolo reconhecido por todos os possíveis clientes.

Uma outra característica de SOA prende-se com a sua capacidade de possibilitar a inclusão de software adicional de forma estática e dinâmica, garantindo desta forma a facilidade de no futuro expandir a arquitectura. Assim, um serviço exterior recentemente publicado pode ser descoberto pela arquitectura e adoptado como um componente para construção de novos sistemas de software que por sua vez podem também ser publicados e disponibilizados como novos serviços.

Desta forma, a arquitectura pretende obter a mínima dependência (*loose coupling*) entre as aplicações (componentes no caso de SOA), permitindo por conseguinte que as mesmas se possam disponibilizar como serviços reutilizáveis.

Por forma a obter serviços exteriores, SOA inclui uma propriedade adicional relacionada com a descoberta (e "lookup") e consequente obtenção dos serviços. Esta propriedade refere-se à habilidade que um cliente possui para dinamicamente descobrir qual o serviço que melhor serve os seus interesses, sendo que por vezes essa responsabilidade é delegada sobre uma aplicação própria para o efeito. No entanto, quando os serviços são descobertos ainda na fase de design da arquitectura, os engenheiros precisam de primeiro descobrir eles próprios os serviços disponíveis e apenas depois decidir se irão reutilizar esses mesmos serviços ou simplesmente desenvolverem eles mesmos novos serviços. No caso de os serviços serem descobertos em "runtime", o processo de descoberta é efectuado automaticamente por uma aplicação de composição e sob a orientação de um fornecedor de serviços (*service provider*) [GL08].

Entre outros benefícios, SOA possui ainda aspectos relevantes tais como a mobilidade de código (devido à estrutura de SOA ser assente em serviços), maior segurança que deriva do

facto de os serviços serem utilizados por múltiplas aplicações e consequentemente herdam os seus próprios mecanismos de segurança, e uma facilidade acrescida na realização de testes devido a serem unitários, ou seja, aplicados a cada serviço individualmente [Stea].

2.3.1 Camada de Integração

A comunicação é efectuada através de mensagens entre serviços, sendo normalmente elaborada com base em protocolos de comunicação e interfaces standard. Entre os diversos protocolos de comunicação envolvidos encontram-se essencialmente Web Services, nomeadamente `SOAP` e `WSDL`, embora soluções de middleware orientado à mensagem (*message-oriented*) como o IBM WebSphere [IBMb] ou serviços de mensagens de Java (`JMS`) [Sun] serem também possíveis. Apesar disto, qualquer tecnologia que possa implementar interfaces de serviço (`WSDL`) directamente e comunicar através de mensagens `XML` pode ser utilizada em `SOA`.

No entanto e apesar das diversas tecnologias, por vezes surgem diferenças entre clientes e servidores a nível de informação ou tecnologia utilizada, criando problemas de interoperabilidade. Nestas situações, duas opções são possíveis. A primeira exige que para cada serviço o módulo criado para o cliente seja construído exactamente conforme as características do serviço que irá invocar. A segunda refere-se à construção de uma camada de integração e de comunicação entre os módulos de clientes e fornecedores [PH07].

A dificuldade inerente à primeira solução, em muito devido à topologia "point-to-point" mantida assim como ao impacto que modificações num determinado serviço possam influenciar outros, levam a que, por forma a garantir maior escalabilidade e integrabilidade de uma solução `SOA`, surjam tecnologias como o *Enterprise Service Bus* (`ESB`) [IBMc].

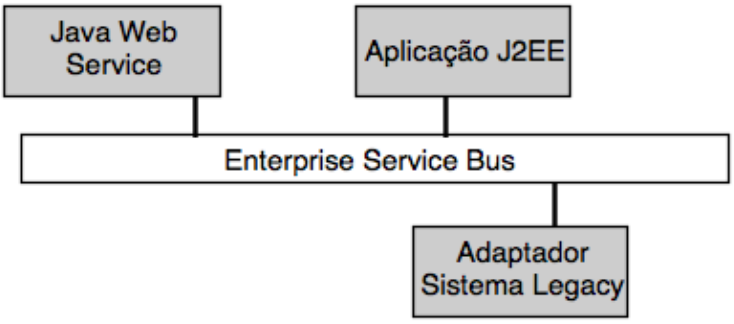

Figura 2.11: Enterprise Service Bus (ESB) numa arquitectura SOA

O `ESB` (ver Figura 2.11) propõe-se como um bloco central para as camadas de abstracção presentes em `SOA`. Proporciona uma forte independência às aplicações através de um sistema distribuído aliado a um sistema de comunicação multi-protocolo. Uma das

suas grandes vantagens surge na forma de *endpoints* lógicos que se atribuem a cada serviço, levando a que modificações ou mesmo substituições de uma aplicação não originem paragens na execução de outras. Assim e devido à independência dos serviços aliada à atribuição de *endpoints* pouco importa qual a plataforma de hardware escolhida ou mesmo a linguagem de programação usada na adição de aplicações [SW08].

A nível de capacidade de manutenção, `SOA`, como uma arquitectura orientada aos serviços, requer uma constante monitorização e controlo de toda a sua comunicação, coordenação e colaboração entre os componentes envolvidos [LZV08].

2.3.2 Qualidade de Serviço

Por vezes, e de forma a aumentar a eficiência de uma solução `SOA`, é adicionada à estrutura `SOA` uma camada exterior de controlo especialmente centrada em aspectos como qualidade, monitorização ou segurança da implementação (ver Figura 2.12). Escolher e desenhar uma arquitectura que satisfaz aspectos funcionais tal como requisitos de qualidade é vital para o sucesso da implementação.

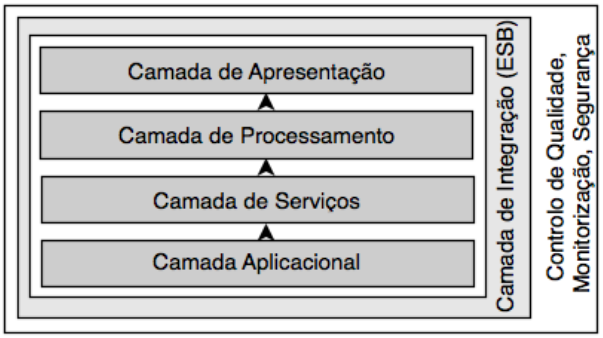

Figura 2.12: Exemplo de solução SOA com políticas de controlo e monitorização [TV08]

Assim, e por forma a controlar aspectos directamente relacionados com a qualidade da solução, certas propriedades ganham especial relevo [OMB07]:

- A *interoperabilidade*, já referida anteriormente, ganha especial relevo na implementação, sobretudo se a solução incluir tecnologias de Web Services. Normalmente a interoperabilidade em `SOA` é sobretudo alcançada através da análise das propriedades sintácticas nos Web Services, nomeadamente através das interfaces `WSDL`, sendo que a nível semântico pouco é ainda conseguido.

- O *desempenho* da solução é também outro aspecto analisável, especialmente centrado nas propriedades da comunicação. Uma dessas propriedades é o *tempo de resposta* de um serviço (tempo necessário para processar um pedido), que pode ser

especialmente condicionado por implementações de serviços compostos (podem utilizar serviços exterior ou sistemas legacy mais lentos), pela própria orquestração dos mesmos, pela invocação dos serviços ou até mesmo pelo acesso aos próprios recursos [OBG08].

Outra propriedade importante na avaliação do desempenho centra-se com o *throughtput* dos serviços, que calcula o número de pedidos processados de acordo com uma unidade temporal. Pode, no caso de ser demasiado elevado, sugerir uma expansibilidade da solução arquitectural.

Outros aspectos ainda relacionados com desempenho referem-se à capacidade de cumprir prazos impostos pelos clientes, à latência nas comunicações ou ao cálculo de um possível processamento excessivo em `XML` (nomeadamente em processos de entrega, parsing, validação e serialização) [DB07].

- *Segurança* em `SOA` engloba propriedades diferentes relativamente a outros modelos arquitecturais. Aspectos críticos como confidencialidade, autenticidade e integridade da informação devem ser considerados. A confidencialidade assegura que o acesso a determinados serviços ou mesmo informação, é apenas disponibilizada a quem possui autorização para tal. A autenticidade pretende garantir que o autor indicado para um serviço é de facto o responsável pela informação disponibilizada. Por fim, a integridade garante que a informação disponibilizada não se encontra corrupta.

  Por forma a incrementar estes parâmetros, devido à crescente adopção de Web Services em SOA, standards como WS-Security ou WS-Trust com base em especificações `SOAP`, `WSDL` e `UDDI` surgiram. No entanto a própria arquitectura continua a necessitar de parâmetros de segurança específicos, onde para tal, paradigmas como Model Driven Security [HB08] ganham forma.

- A *confiabilidade* em sistemas `SOA` relaciona-se com a capacidade do sistema ser capaz de permanecer operacional sem a existência de falhas. Em `SOA`, a confiabilidade pode ser directamente relacionada com dois tipos: em mensagens e em serviços. A confiabilidade relativa às mensagens centra-se essencialmente em falhas que as conexões podem sofrer no processo de entrega de uma mensagem ou na eventualidade da entrega ser efectuada na ordem errada. Normalmente tal é parcialmente resolvido através de middlewares específicos para as comunicações. Já a confiabilidade nos serviços é referente aos serviços e à capacidade destes operarem correctamente e sem falhas. Normalmente o principal problema centra-se com a preservação da

informação durante falhas e acesso concorrente [OMB07].

- *Disponibilidade* em SOA mede o tempo total em que um sistema, ou componente, se encontra operacional e acessível quando requisitado para utilização. Em SOA a disponibilidade é uma preocupação partilhada pelo utilizador do serviço assim como pelo próprio fornecedor. Do ponto de vista do utilizador, se o serviço não se encontrar acessível, então o sistema não preenche os seus requisitos funcionais. No caso do fornecedor, para o seu serviço ser utilizado, então este terá que se encontrar disponível. Assim e de maneira a evitar tempo de paragem, soluções relacionadas com a infra-estrutura, tais como replicação ou sistemas de balanceamento de carga, são por vezes utilizadas.

- Outro aspecto qualitativo em SOA compreende a *capacidade do sistema ser modificado*. Relaciona-se com a capacidade da arquitectura ser facilmente modificada e a baixo custo. Normalmente, e de forma a tentar alcançar uma alta capacidade de modificação, os serviços presentes numa solução SOA possuem pouca interdependência possibilitando assim uma maior facilidade nas alterações sem danificar outras implementações. No entanto, a modificação de interfaces de serviços pode ser mais crítica se se encontrarem implementadas por outras aplicações. Em adição, a capacidade de "extender" a solução sem que tal afecte outros componentes é também uma característica importante na análise da capacidade da solução poder ser modificada. Um caso específico referente a este aspecto é por exemplo a adição de novos serviços.

- *Testabilidade* refere-se à facilidade com que a execução de testes e critérios a si associados são efectuados. Em SOA, o processo de realizar testes na arquitectura pode tornar-se complexo devido a razões como a não disponibilidade do código fonte de serviços exteriores ou porque a descoberta de serviços é efectuada apenas em *runtime*, tornando-se assim quase impossível prever qual o serviço que será utilizado.

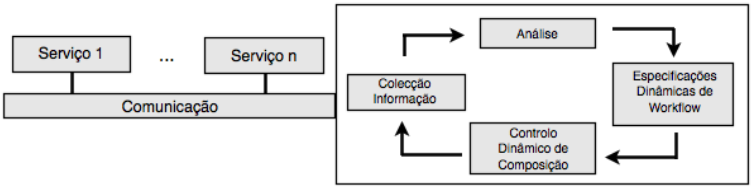

Figura 2.13: Factores representativos de testabilidade em Serviços [TGWC06]

No cálculo de testabilidade, um centro de controlo efectua testes às comunicações,

normalmente indexado à camada de comunicações como ilustrado na figura 2.13. O bloco da direita representa um outro centro de controlo contendo normalmente quatro blocos funcionais distintos por forma a realizar uma variedade de tarefas relacionadas com análise, informação existente, controlo de composições em serviços ou especificações de workflow [TFC+06].

- Por fim, outro aspecto relacionado com a qualidade refere-se à *usabilidade* em soluções SOA. Usabilidade pode ser definida como o processo recorrente da medição de qualidade atribuída ao nível de experiência do utilizador com os serviços [OMB07]. Assim, em SOA, uma demora no processo de uma chamada a um serviço (sobretudo se o serviço for exterior) pode prejudicar a usabilidade. Para tal é sugerido que a comunicação seja separada da aplicação contendo o utilizador do serviço, providenciando assim a transmissão de feedback mais rapidamente ao utilizador. Por vezes, a usabilidade é medida com recurso a técnicas HCI normalmente efectuadas sobre as interfaces gráficas acessíveis aos utilizadores, onde algumas dessas técnicas envolvem planeamentos cognitivos ou mesmo "Think Aloud Protocol Evaluation" [BMS+08].

2.4 Software as a Service

Em 1999, o Pennine Group[4], previu que o desenvolvimento de novos estilos arquitecturais com base em modelos por objectos, ou através da utilização de componentes, sofreria uma estagnação. Ao invés, previam um futuro no qual a forma como o software seria entregue aos utilizadores iria ser totalmente diferente [TBB03]. O conceito de Software as a Service pode ser visto como uma concretização dessa previsão.

Software as a Service (SaaS) é um conceito de disponibilização de software que, na sua essência separa a posse do software do utilizador, ou seja, o proprietário do software passa a ser um vendedor que o implementa e permite que o utilizador o execute a pedido através de alguma forma de cliente via internet ou intranet [LZV08]. De facto, é previsível que o SaaS origine um impacto na maioria dos departamentos TI empresariais originando assim não só uma mudança na forma como software é entregue mas também na forma como este é construído, vendido, comprado e utilizado. Deste modo, é esperado que SaaS se confirme como uma opção comum para soluções de software [Nat07].

Este modelo propõe aos utilizadores que o software lhe seja disponibilizado como um serviço que é pago dependendo do seu grau de utilização, similar assim a serviços do quotidiano como por exemplo a utilização de electricidade. Deste modo, este modelo, tal como

[4]consórcio formado por engenheiros de software de várias universidades britânicas

um conjunto de serviços distribuídos, que podem ser configurados e integrados aquando da entrega, pode ultrapassar certas limitações actuais relacionadas com a utilização do software, a sua instalação e mesmo a sua própria evolução [TBB03].

Um exemplo concreto de `SaaS` são os populares sistemas de e-mail disponíveis na Internet, tais como o Google Mail ou o Microsoft Hotmail. De facto, cada um destes sistemas são alojados no seu proprietário, neste caso a Google e a Microsoft, que são responsáveis pelo funcionamento da aplicação. Os utilizadores por sua vez podem aceder à sua conta através de uma aplicação cliente, como por exemplo um browser. No entanto todos os dados encontram-se do lado do proprietário e este apenas os fornece quando o software cliente é executado tal como defende o modelo `SaaS`.

Assim, este modo de entrega implica obrigatoriamente um modo de implementação diferente e por conseguinte um modelo arquitectural distinto.

Normalmente, por forma a disponibilizarem as suas aplicações pela Web, fornecedores recorriam ao modelo presente num *Application Server Provider* (`ASP`) como forma de obterem algo semelhante a `SaaS`. A arquitectura de um `ASP` representa uma forma de entregar software a pedido e por conseguinte pode ser considerada como pseudo-`SaaS`. De facto, o modelo `ASP` representa uma arquitectura onde cada aplicação é de certa forma singular, ou seja, cada aplicação é construída com base num determinado cliente e será totalmente exclusiva a esse cliente tornando-se uma aplicação independente. Desta forma, a arquitectura de um `ASP` torna-se essencialmente um serviço de armazenamento onde os clientes possuem os seus servidores privados e as suas aplicações são armazenadas numa espécie de fornecedor de serviços [LT08].

No entanto, as arquitecturas `SaaS` de hoje apresentam algumas diferenças nomeadamente ao nível de utilização, ou seja, enquanto que com um `ASP` as aplicações são singulares, em `SaaS` as aplicações são *multitenant*[5]. Deste modo, a uma aplicação em `SaaS` podem ser adicionadas modificações ou mesmo extensões face ao requisitado por um cliente anterior, tornando-a assim disponível para este simultaneamente. De facto, é expectável que as aplicações `SaaS` tirem partido de benefícios actuais, como centralização através de uma única instância para vários clientes e serem capazes de fornecer algo competitivo com as aplicações tradicionais.

Assim, e por forma a obter estas propriedades, tais como eficiência multi-utilizador, expansibilidade ou mesmo configurabilidade por parte de alguns utilizadores, `SaaS` distingue-se de um ASP ao incorporar uma camada de serviços como visível na Figura 2.14.

[5]tenant - empresa cliente que utiliza uma aplicação SaaS

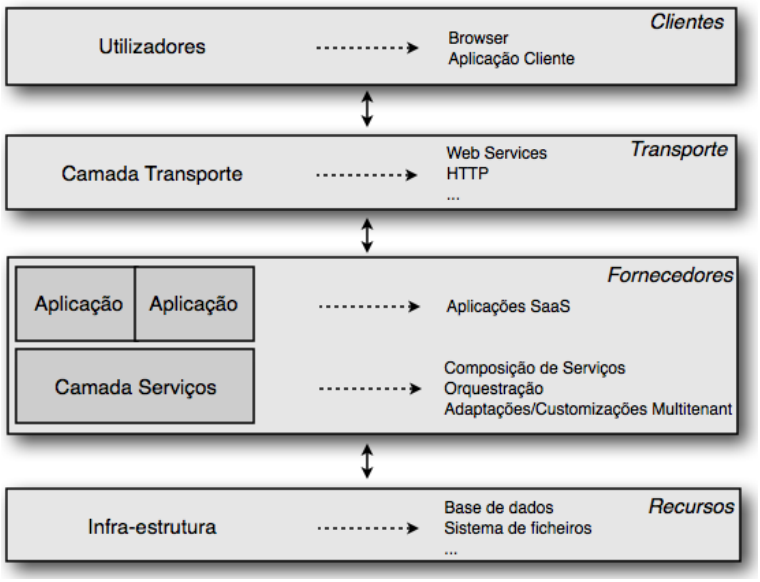

Figura 2.14: Exemplo de uma Arquitectura SaaS

Na Figura 2.14 pode-se constatar que o fornecedor de `SaaS` armazena a aplicação e a informação centralmente, podendo assim efectuar upgrades à aplicação de forma transparente, e ser capaz de providenciar acesso aos seus clientes pela Internet através de um browser ou de uma aplicação cliente. Fornecedores providenciam `APIs` que expõem a informação das aplicações e sua funcionalidade de modo que programadores a possam utilizar na criação de aplicações compostas (com Web Services). Por vezes, para além das `APIs`, os fornecedores providenciam ainda ferramentas que permitam aos seus clientes modificar a própria informação armazenada, fluxos de informação e outros aspectos que necessitem para a utilização das aplicações, utilizando mecanismos de segurança próprios para manter a informação transmitida segura.

De salientar que apesar de na imagem apresentada, os recursos se encontrarem separados da plataforma principal, muitas vezes, os fornecedores optam por incluí-los nessa mesma plataforma por forma a obterem um maior controlo sobre os dados. No caso ilustrado pela Figura 2.14, no entanto, os recursos são acedidos pela plataforma através de Web Services configurados para o efeito.

2.4.1 Metodologia de desenvolvimento de aplicações em SaaS

As diferenças arquitecturais e do modelo defendido por `SaaS` influenciam o ciclo de vida inerente às aplicações sobretudo devido ao seu modelo de entrega. Este por sua vez causa implicações na própria metodologia de desenvolvimento de software comparativamente ao modelo tradicional. Esta metodologia é apresentada na Figura 2.15 com base no modelo

iterativo para desenvolvimento de aplicações.

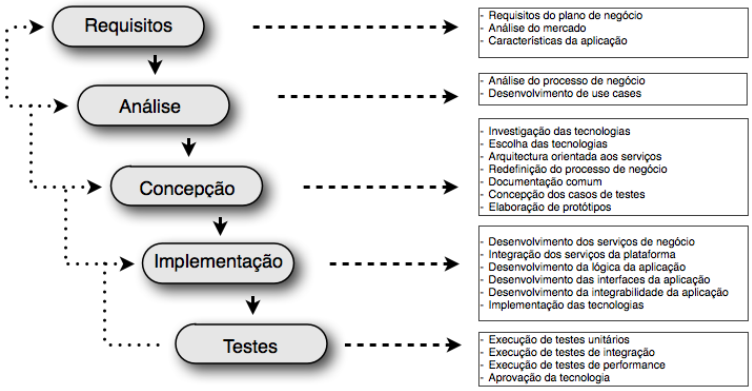

Figura 2.15: Principais linhas de desenvolvimento de aplicações em SaaS [ECM08]

Apesar do processo de desenvolvimento apresentado possuir fases semelhantes ao modelo tradicional (em cascata) tais como análise de requisitos, concepção ou implementação, as implicações de cada uma são bastante distintas. Mesmo relativamente às linhas orientadoras tais como o desenvolvimento das interfaces da aplicação, o desenvolvimento da lógica da aplicação, ou por exemplo a concepção dos casos de teste, o modo de implementação das mesmas será diferente tendo obrigatoriamente que ser refinado por forma a reflectir os requisitos inerentes ao novo modelo de entrega de software. Para além deste refinamento, surgem ainda novas linhas orientadoras criadas devido à diferença arquitectural imposta pelo `SaaS`.

De facto, através da análise das principais fases de desenvolvimento, podemos facilmente distinguir o que seria tradicionalmente efectuado numa fase e o que terá que ser adicionado em `SaaS`: [ECM08]

- durante a fase de levantamento de *requisitos*, sobretudo devido ao estilo distribuído da arquitectura e consequente utilização da aplicação por um elevado número de subscritores, `SaaS` implicará automaticamente a introdução de outros requisitos não-funcionais tais como concorrência, escalabilidade ou desempenho no armazenamento entre outros.

- a *análise* por sua vez, terá que ser efectuada de acordo com a perspectiva de negócio do cliente que por sua vez irá utilizar alguns serviços disponibilizados pela arquitectura por forma a compor essa mesma lógica, originando a que deste modo a arquitectura consiga satisfazer uma vasta gama de clientes. No entanto, e para além deste ponto, a análise em `SaaS` passará também obrigatoriamente pelo levantamento das capacidades e limitações da plataforma.

- já a *concepção* consiste essencialmente na elaboração de documentação com vista à implementação. Assim, e para além da documentação comum, a restante documentação incorpora essencialmente a construção de modelos `UML`, o levantamento e posterior definição das tecnologias a serem utilizadas, a elaboração de casos de testes ou mesmo protótipos tecnológicos e por fim a concepção dos componentes aplicativos com vista à criação de um infra-estrutura por serviços.

- a fase de *implementação* em `SaaS` diferencia-se do método tradicional devido à necessidade de incluir também os processos na plataforma. Assim, são implementadas as lógicas de negócio da aplicação assim como os próprios serviços inerentes à plataforma. No entanto, e para além do desenvolvimento da aplicação, nesta fase são também implementados todos os mecanismos necessários relativamente à integrabilidade da aplicação na plataforma `SaaS`. De facto, e por forma a obter uma maior integração de uma aplicação `SaaS` com um determinado *backend* ou outras aplicações `SaaS`, são utilizados padrões com baseados em *Enterprise Application Integration* (`EAI`) durante a customização da aplicação para um determinado cliente [SML08].

- Por fim, a fase de *testes* verifica possíveis erros na aplicação sendo que em `SaaS` a maior diferença centra-se com a execução de testes de integração com vista a poder validar a interacção correcta entre as aplicações e a plataforma.

Não é apenas durante o processo de desenvolvimento de uma aplicação que se notam diferenças entre os modelos apresentados mas durante todo o ciclo de vida da aplicação. Aspectos relacionados com a comercialização, a utilização e principalmente a capacidade de manutenção são também afectados pelas mudanças arquitecturais.

Na tabela 2.1, é estabelecida uma comparação relativamente às características presentes no ciclo de vida de uma aplicação de software entre o modelo tradicionalmente utilizado e o modelo em `SaaS`. São notórias as diferenças entre os modelos, não só durante o processo de desenvolvimento da aplicação mas durante todo o seu ciclo de vida. No entanto, para além dos aspectos relacionados com a aplicação em si as diferenças realçam também o que `SaaS` representa para os seus clientes e fornecedores.

De facto, através da utilização de `SaaS` alteram-se procedimentos tradicionais como a propriedade do software e as responsabilidades inerentes à manutenção do software e da infra-estrutura que passam dos utilizadores para os fornecedores. Para além disto, os custos de software são reduzidos de forma substancial essencialmente através da sua especialização (derivada do cliente), abrindo assim caminho a novas oportunidades de negócio sobretudo na área das pequenas e médias empresas.

Fase	**Tradicional**	**Serviço**
Concepção e Especificação de Software	Documentos especificativos tem elevado grau de importância. A especificação não é influenciada pelo modelo de negócio. Designers do software apenas se preocupam com a especificação do mesmo.	Prototipagem pode ser facilmente utilizada devido à habilidade de poderem ser efectuadas entregas prematuras da aplicação. Requisitos são definidos apenas pelo mercado alvo. A concepção do software é muito influenciada pelo modelo de plataforma orientada aos serviços.
Construção de Software	A implementação é algo influenciada por bibliotecas ou frameworks. Programadores confiam essencialmente nas linguagens programacionais, bibliotecas e frameworks. Componentes externos são opcionais.	Plataforma afecta directamente a arquitectura do sistema. Programadores têm obrigatoriamente que conhecer a `API` da plataforma. Componentes partilháveis como autenticação ou de logging são providenciados pela plataforma.
Deployment de Software	Um deployment por cliente. Normalmente o deployment é um processo customizável dependendo do ambiente de execução.	Apenas um deployment por fornecedor de serviço. Processo de deployment standard.
Comercialização do Software	Não participa no processo de comercialização.	O modelo de negócio do proprietário pode ser publicitado pelo próprio software ou através da plataforma.
Utilização do Software	Através de instalações locais ou por `LAN` ou `WAN`.	Acesso através da Internet causa mais preocupações e obriga a esforço extra devido a aspectos como largura de banda ou experiência de utilização.
Manutenção do Software	Actualizações efectuadas pelo próprio cliente.	Actualizações do software assim como da plataforma só podem ser efectuados pelo proprietário. Downtime de certas aplicações podem afectar a base de clientes total.

Tabela 2.1: Diferenças no processo de desenvolvimento de software entre o modelo tradicional e o modelo orientado a plataformas de serviços [ECM08]

Normalmente, uma aplicação típica `SaaS` é oferecida aos clientes, ou directamente pelo vendedor, ou através de um grupo intermediário designado agregador, que compõe ofertas `SaaS` oriundas de diferentes vendedores e as oferece como uma aplicação unificada. Relativamente ao método de pagamento em `SaaS`, surgem algumas vantagens relativamente ao modelo tradicional de fornecer software quer ao nível da empresa proprietária do software, quer ao nível do próprio utilizador. Na perspectiva do proprietário, surgem vantagens como [LT08]:

- *redução dos custos na entrega do software.* Devido ao modo de entrega de `SaaS`, custos relacionados com a comercialização de software são assim reduzidos.

- *expansão na aplicabilidade dos seus produtos.* Pequenas e médias empresas, antes impossibilitadas de comprar licenças de software tornam-se agora possíveis clientes em muito devido a uma acrescida comunicação entre utilizadores e proprietários do software.

- *aumento de competitividade entre fornecedores de software.* Reduções no preço de software assim como na forma como os lucros são efectuados obrigam a uma de maior rapidez no lançamento do software assim como uma tentativa de rapidamente acrescentar novas características ao produto, incrementando a competitividade.

- *evitar a cópia ilegal de software.* Como resultado do modo como `SaaS` é desenvolvido, todas as aplicações, autenticações ou mesmo upgrades do software encontram-se do lado dos proprietários reduzindo assim a cópia ilegal do software.

Na perspectiva do utilizador as vantagens surgem essencialmente a nível da especialização obtida por SaaS através de aspectos tais como:

- *redução de custos de informatização em pequenas e médias empresas.* O modelo tradicional de software empresarial implica custos iniciais elevados que em adição aos custos diários de manutenção e custos associados a profissionais contratados para a informatização os tornam por vezes insustentáveis. `SaaS` contrasta esta visão, ao ser responsável por toda a implementação e futura capacidade de manutenção.

- *maior focalização no "core business".* Através de `SaaS`, as aplicações podem ser "encomendadas" pelos utilizadores de acordo com as suas preferências, originando a que desta forma os módulos pedidos possam sofrer ajustes por forma a se adaptarem ao "core business" da empresa.

- *melhor serviço prestado.* Para além da maior flexibilidade na utilização dos serviços, através de feedback providenciado aos proprietários do serviço, as aplicações podem ser actualizadas mais rapidamente por forma a satisfazer os utilizadores.

Capítulo 3

Trabalho Relacionado

Neste capítulo serão retratados aspectos que se propõem a simplificar a construção de aplicações baseadas na obtenção de conteúdos através de serviços Web. O seu objectivo centra-se fundamentalmente em proporcionar aos seus clientes o rápido acesso aos serviços, através de middlewares genéricos construídos para o efeito. Neste sentido, serão descritas algumas soluções baseadas na elaboração de camadas de abstracção orientadas à construção de Mashups e criação de directórios orientados à catalogação de serviços, encontradas na literatura.

3.1 Ferramentas de Desenvolvimento de Mashups

Presentemente, o desenvolvimento de mashups ainda não é um processo célere e simples, de acordo com a própria natureza dos mesmos. De facto, ainda hoje, a grande dificuldade patente na construção de mashups centra-se na elaboração de interfaces gráficas e integrabilidade das fontes de dados.

Assim, e devido a uma necessidade patente na indústria de tentar acelerar estes processos, algumas soluções foram surgindo sob a forma de ferramentas programáticas de forma a produzir um modo simples e mais fácil de desenvolver mashups.

Assim, e focalizando-se essencialmente nos problemas anteriores, estas ferramentas na sua maioria propõem-se a resolver um problema comum, o encapsulamento da componente de comunicações com os Web-Services. Deste modo, e de forma a retirar esta responsabilidade ao utilizador, as ferramentas proporcionam uma camada de abstracção integradora capaz de gerir toda a comunicação com os serviços assim como garantir a integrabilidade de outros que possam vir a ser incorporados. Adicionalmente, algumas das ferramentas procuram ser capazes de gerir automaticamente os módulos de integração dos serviços web de forma a que a própria selecção dos serviços seja de certa forma transparente para

o utilizador.

Desta forma, é ambicionado que para o utilizador não seja obrigatório possuir conhecimentos programáticos avançados para desenvolver um mashup originando que, graças a esta integrabilidade, seja esperado que o espectro de possíveis criadores de mashups cresça de forma acentuada.

Uma dessas ferramentas designa-se por Popfly[Mica]. Desenvolvida pela Microsoft, o Popfly funciona com base em componentes, designados por blocos, através do encapsulamento de cada Web Service como um bloco reutilizável. Cada bloco possui operações relacionadas com inputs e outputs especificadas através de um descritor XML sendo possível a utilização destes blocos como forma de obter a interface final.

De facto, a construção dos próprios blocos é efectuada através de um wizard presente na ferramenta sendo que o utilizador pode usar um bloco já existente como base para a obtenção de novos. Desta forma, é garantido que o espectro de serviços possíveis não é limitado ficando dependente apenas dos utilizadores.

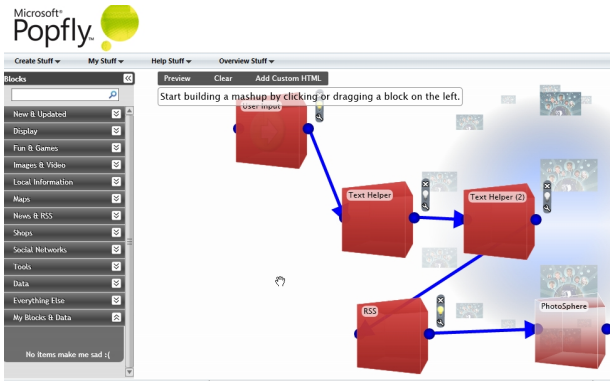

Figura 3.1: Exemplo de Execução do Microsoft Popfly

Através de um ambiente visual acessível, como é visível na Figura 3.1, aos utilizadores bastar-lhes-á seleccionar os blocos pretendidos, ligá-los entre si e conceber a interface, podendo para tal utilizar tecnologias como `AJAX` [Gar], `DHTML` [W3S], ou Silverlight (`XAML`) [Corb], sendo que toda a complexidade relacionada com a obtenção e integração da informação será encapsulada.

A Yahoo por sua vez criou uma ferramenta que permite a junção de feeds de informação por forma a criar mashups de dados através de um editor visual. Designado por Yahoo Pipes[Yah], esta ferramenta assenta sobre uma utilização de operadores que conectam as fontes de informação dispostas por módulos que obtém a sua informação através de Web Services.

Assim, cada aplicação resultante é designada por Pipe, consistindo essencialmente na união entre dois ou mais módulos, cada um com a sua função específica, e na ligação

de operadores tendo por base tarefas como a manipulação da informação, utilização de expressões regulares ou mesmo extracção de dados.

A nível do editor visual, o Pipes é semelhante ao Popfly, permitindo aos utilizadores a ligação entre os módulos através de ligações entre eles, utilizando para tal setas entre os módulos para especificar os seus inputs e outputs constituindo assim o fluxo de informação da aplicação. Por outro lado, e apesar destas facilidades, programadores mais avançados poderão não gostar de poderem apenas manipular a representação visual do mashup e consequentemente não lhes ser facultado o acesso ao código fonte.

O Google mashup Editor[Goob], agora designado por App Engine[Gooa] desenvolvido pela Google, consiste num ambiente de desenvolvimento com base em templates. Similarmente às aplicações anteriores, o modo de funcionamento da aplicação assenta sobre a combinação de módulos standard que possuem como principais objectivos permitir aos utilizadores o encapsulamento e estruturação dos dados exteriores.

Assim, criar mashups no GME depreende a necessidade do utilizador desenvolver templates para interfaces gráficas através de uma composição de elementos de layout em `XML` e `HTML`/`CSS` com código JavaScript. Deste modo, a capacidade principal desta ferramenta centra-se com a sua capacidade de em tempo de execução, preencher esses mesmos templates constituindo assim páginas Web prontas a serem utilizadas sem qualquer interacção por parte do programador.

A Intel através da sua ferramenta, Intel Mash Maker[Int, EG07] disponibiliza um ambiente de integração de informação com base em anotações dispostas sobre páginas Web através de um plug-in bastante poderoso para o browser.

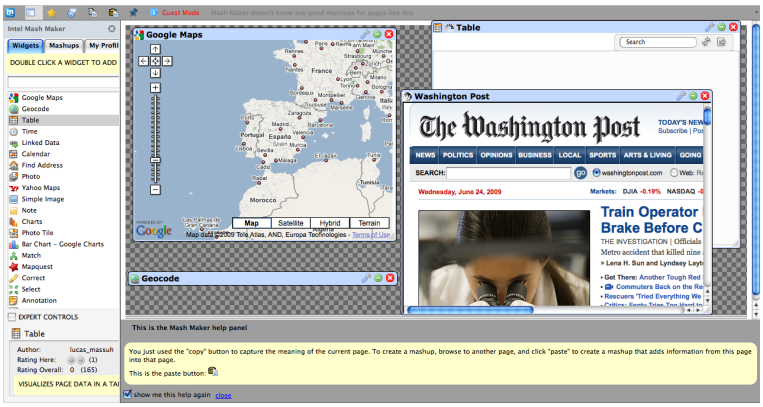

Figura 3.2: Ambiente de Execução do Intel Mash Maker

Deste modo, e para além de poder obter os seus inputs via widgets ou `RSS` feeds, o Mash Maker permite aos utilizadores anotarem páginas Web enquanto navegam e poderem posteriormente extrair tais conteúdos das páginas anotadas. Deste modo e como visível

na Figura 3.2, compôr mashups via Mash Maker ocorre através de um modelo de *copiar-colar*, sendo que um utilizador poderá optar por introduzir o conteúdo de uma página directamente noutra como cabeçalho, ou poderá combinar os conteúdos das duas com base nas anotações, fundindo-os numa só página.

O QEDWiki[IBMa], agora renomeado Lotus mashups, é a ferramenta da IBM assente sobretudo num estilo Wiki e vocacionada principalmente para programadores profissionais.

A sua base centra-se na criação de comandos ou widgets por parte dos utilizadores, que são basicamente aplicações simples, ou partes de código passíveis de serem utilizados, com base em JavaScript ou `PHP`. De seguida, estes comandos podem ser compostos em aplicações completas através de um ambiente visual onde os programadores podem configurar interactivamente os seus widgets, resultando por conseguinte em reduções de tempo de desenvolvimento. Assim, e por forma a constituir um mashup, bastará aos programadores seleccionar um layout para a página Web final, e compor os widgets, ligando-os entre si, consoante o propósito final.

David Maier et al propõem em [MMD06] uma ferramenta designada por Mash-o-Matic capaz de extrair, limpar, e combinar diferentes fragmentos de informação e automaticamente gerar dados passíveis de serem utilizados em mashups ou mesmo gerar os próprios mashups.

Assim, e através da utilização de XML e tecnologias associadas tais como `XSLT` ou `XPath`, a informação é guardada através de um documento anexo, o Sidepad, por forma a ser coleccionada e organizada, e posteriormente processada através de um motor de queries multi-nível por forma a preencher os requisitos do mashup. Desta forma a informação será transformada nos formatos pretendidos para o mashup, sendo a criação do mashup finalizada por intermédio de um processo auxiliar com a missão de constituir o código final.

Em [MRT07] é proposta uma ferramenta com base em *Domain-Specific Languages* (`DSL`)[1], o Swashup, centrada sobre uma lógica segundo a qual dois paradigmas actuais da Web, a proliferação de `API`s públicas ou `RSS` Feeds assim como a crescente utilização de linguagens de programação dinâmicas para construção de aplicações Web, podem ser complementares em muito devido a uma visão de uma Web programável.

Assim, é proposta uma `DSL` por forma a representar as interfaces comuns presentes nos serviços, assim como ser capaz de providenciar um modelo uniforme para as operações e interacções entre os serviços. Adicionalmente, os autores apresentam a combinação desta `DSL` com uma plataforma baseada em Ruby on Rails, permitindo assim um alto nível de

[1] Linguagens criadas de forma a resolver problemas específicos

abstracção.

Assim, é possibilitado aos utilizadores uma representação explicita das actividades envolvidas no mashup que podem posteriormente ser integrada na plataforma RoR permitindo posteriormente a adição de páginas Web incluindo AJAX ao mashup por forma a elaborar a interface visual do mashup.

3.2 Directório de Serviços

As tecnologias de Web Services tornaram-se na última década bastante populares em muito devido à sua grande potencialidade em diversas áreas de aplicação. Deste modo, surgiu uma necessidade de agrupar esses mesmos serviços por forma a serem disponibilizados, consoante a sua aplicabilidade, da melhor forma possível. Assim, uma área de desenvolvimento bastante activa com relação directa com Web Services, são os serviços de directoria.

Um directório de serviços é por norma, um local centralizado, cuja funcionalidade principal é facilitar a procura de serviços por clientes, assim como ser um local onde programadores possam publicar interfaces para os seus serviços por forma a obterem visibilidade para os mesmos. Na maioria dos casos, estes directórios apresentam-se como uma infra-estrutura adicional, ao estilo de SOA, onde os clientes podem obter um serviço dependendo da metodologia adoptada pelo directório.

De facto, no modelo original de Web Services, foi proposto um serviço semelhante, o UDDI como forma de os serviços poderem ser agrupados e disponibilizados dinamicamente consoante a sua necessidade. No entanto, e devido a alguns lacunas patentes no UDDI, diversas alternativas surgiram por forma a dar resposta a esta necessidade.

Em [DVV03] é apresentada a noção de contexto como possível abstracção para um serviço. São definidas categorias de serviço , como por exemplo, fotos ou video, às quais será atribuído um contexto. Desta forma, o autor pretende optimizar a procura por determinados serviços ao excluir categorias de acordo com o utilizador. Assim, os contextos serão incluídos num grafo MOEMMultidimensional Object Exchange Model[2] [GG98] dependendo da sua área de uma forma semântica, atribuindo lógica à estrutura do grafo. Os serviços por sua vez, tornar-se-ão folhas do grafo de acordo com o seu contexto. A Figura 3.3 apresenta um exemplo da utilização do grafo para representar o directório.

O grafo, por sua vez incluirá um algoritmo próprio de navegação que permitirá optimizar a procura. A selecção de um serviço será efectuada com base na *query* do utilizador, e

[2]grafo orientado para representação multidimensional de informação semi-estruturada

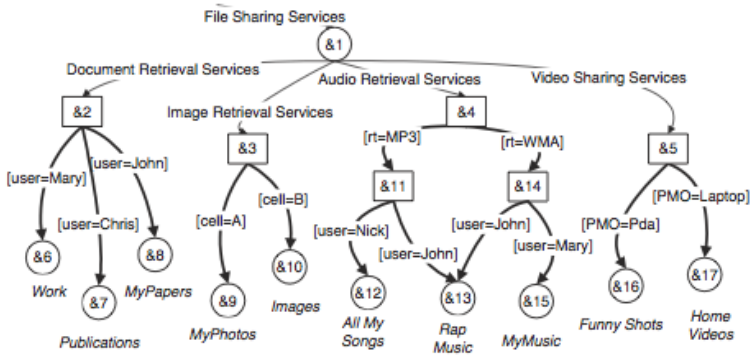

Figura 3.3: Representação de um Directório de Serviços por intermédio de um grafo [DVV03]

em requisitos não-funcionais adicionados a cada folha, como por exemplo a disponibilidade (através da medição do tempo total que o serviço está online).

Shuping Ran aponta em [Ran03] a lacuna do `UDDI` face a requisitos não-funcionais propondo uma solução tendo por base um novo modelo de registo de descoberta para Web Services. Entre alguns dos requisitos não-funcionais, o autor atenta a aspectos centrados em ambiente de execução como desempenho ou fiabilidade, em suporte à transacção como integridade, ou em segurança como confidencialidade.

Assim, é proposto que para além das entidades normalmente envolvidas no processo de execução de um Web Service, seja adicionada uma nova entidade, o `QoS` Certifier, que funcionará paralelamente com o `UDDI`, tendo as funções exclusivas de verificar parâmetros `QoS` quando solicitado pelo cliente do serviço, e ainda a de certificar esses mesmos parâmetros no acto de publicação de um serviço no `UDDI`.

Deste modo, o autor pretende garantir que um serviço é apenas publicado no registo `UDDI` se passar todos os testes de `QoS`, emitindo posteriormente um certificado de qualidade relativamente ao serviço. Assim, a procura efectuada no registo terá como resultados, para além do `WSDL` do serviço encontrado, informação `QoS` acessível aos futuros clientes.

Em [KFS+06] é proposta uma framework para descoberta de serviços orientada à arquitectura. Assim, é adoptada uma metodologia na qual o processo de descoberta se baseia no constante desenvolvimento da arquitectura.

O processo tem por base a concepção de uma framework a partir de dois módulos principais, onde o primeiro se direcciona para a integração de modelos UML2.0 enquanto o segundo se centra na execução de queries. Deste modo, o workflow envolvido executa por intermédio do módulo `UML`, a integração de serviços candidatos para os modelos, assim como o levantamento de queries centradas nas funcionalidades do serviço, suas propriedades e condições apartir dos modelos de design. De seguida, e com base no

output do passo anterior, o motor de queries executa as queries através de pesquisas em diferentes registos de serviços.

Devido a tratar-se de uma arquitectura de serviços centralizada, o autor sugere que este processo constante pode assim para além de originar novas iterações levar a que novos modelos de design implementados possam ser desde logo identificados.

Gokay Akkus em [Akk07] aponta a necessidade de sistemas dinâmicos e adaptativos por forma a que alterações súbitas não afectem a composição dos serviços. Como tal, propõe um sistema baseado em agentes que guie a descoberta dos serviços e composição dos mesmos através da integração de grafos em web services. Este sistema, defende a utilização de aspectos semânticos na categorização dos serviços por forma a resolver a dificuldade actual de automaticamente serviços poderem ser descobertos, compostos e mesmo executados.

É proposto que a cada agente seja associado um grafo onde a cada nodo será associado um serviço que possuirá um peso. Posteriormente, e de acordo com a especificação do cliente, o agente calculará qual o serviço mais indicado recorrendo a um algoritmo para cálculo do caminho mais curto para navegação no grafo.

Capítulo 4

Concepção do Sistema de Mediação

O objectivo principal desta tese centra-se na concepção e desenvolvimento de um middleware capaz de encapsular serviços externos e disponibilizar aos clientes o acesso uniforme aos mesmos, independentemente de qual, ou quais, é que estão a ser utilizados.

Seguindo um plano de desenvolvimento aproximado à concepção de aplicações para `SaaS`, neste capítulo será feita uma análise com vista à definição de uma metodologia para a construção de um sistema de mediação. Deste modo, serão abordados aspectos envolvidos na concepção do sistema, sendo realçados alguns particularmente importantes de forma a serem explicadas as decisões tomadas na elaboração do middleware.

4.1 Requisitos

Considere-se uma situação em que um programador de uma aplicação web precisa de incluir nas suas páginas informação sobre localizações em mapas. Actualmente, são vários os fornecedores desde tipo de serviço e outros podem surgir no futuro. O programador ou opta por um dos fornecedores, estando condicionado pela sua disponibilidade e desempenho, ou inclui no seu código mecanismos de escolha do melhor fornecedor em cada momento.

O objectivo principal de um sistema de mediação centra-se precisamente em disponibilizar a aplicações web (entre as quais mashups) um mediador capaz de sintetizar diversos fornecedores de serviços através de uma abstracção, retirando a necessidade dos programadores necessitarem de compreender e analisar diversas `APIs` em momentos de implementações de serviços e simultaneamente relegando para o mediador aspectos relacionados com falhas de serviços ou políticas de melhor fornecedor.

Assim, um dos requisitos do sistema de mediação depreende-se com a capacidade do sistema ser capaz de fornecer determinado tipo de serviço com base em parâmetros

relativos à sua qualidade de serviço.

O sistema deve ainda ser capaz de determinar alternativas viáveis para o utilizador na ocorrência de falhas por parte de um fornecedor, disponibilizando-lhe um outro fornecedor capaz de exercer as mesmas funcionalidades do inicial, mas com a particularidade de que tal operação seja completamente transparente para a aplicação cliente.

Simultaneamente, o sistema de mediação deve ser capaz de fornecer ao cliente um fornecedor elegido pelo próprio se ele assim o entender, assim como ser capaz de interpretar características funcionais do serviço pretendido pelo utilizador.

Para além destas funcionalidades, foi compreendido durante a concepção a necessidade do sistema ter por base algumas características obrigatórias, qualificáveis como requisitos não-funcionais, tais como integrabilidade, modularidade, facilidade de utilização e expansibilidade.

Na continuação deste capítulo serão explicados todos os passos envolvidos na concepção do sistema de mediação, realçando alguns particularmente críticos para a sua elaboração.

4.2 Análise

Com base nos requisitos descritos na secção anterior é possível desde logo definir alguns tópicos de análise directamente relacionados com a concepção de um mediador. Tópicos como perceber o que pode conter um pedido, como implementar os diversos fornecedores ou até mesmo como efectuar a monitorização dos serviços estarão obrigatoriamente ligados à forma como o mediador se comportará. Adicionalmente, o mediador terá ainda que ser concebido de forma a garantir uma alta integrabilidade com fornecedores de serviços de forma a que a sua expansibilidade esteja assegurada.

A invocação do sistema como um serviço web é também um aspecto importante. Na verdade, o facto do sistema ser disponibilizado desta forma permite desde logo perceber que a sua lógica de execução poderá conter elementos tipicamente encontrados em serviços comuns.

Por norma, um serviço web possui uma lógica que pode diferir comparativamente a outros serviços (i.e. na sua implementação) mas que normalmente se baseia em quatro fases de execução:

- Pedido/Invocação - um pedido é efectuado ao fornecedor do serviço pelo cliente através de uma operação disponível na `API` do serviço.

- Análise/Interpretação - o servidor recebe o pedido e procura efectuar a sua interpretação.
- Business/Execução - o fornecedor de serviço procura construir uma resposta para o pedido com base na sua lógica de negócio.
- Resposta - o fornecedor devolve os resultados ao cliente.

Esta estruturação está patente em todos os web services existentes. No caso de uma plataforma de mediação, fases como a recepção do pedido, a sua análise e até mesmo a devolução da resposta serão semelhantes e como tal estarão também presentes. No entanto, a execução do serviço será bastante diferente devido essencialmente a dois factores.

O primeiro factor provém dos principais requisitos do mediador, relacionados com a sua capacidade em agrupar sobre si vários fornecedores de modo a que seja possível eleger um deles na sequência de um pedido. Como tal, é necessário que o mediador possua um mecanismo capaz de seleccionar um desses fornecedores com base em determinados aspectos que podem surgir por indicação do cliente ou por análise aos próprios serviços.

O outro factor centra-se precisamente com a lógica de negócio do sistema de mediação. O sistema não pretende resolver os pedidos a si efectuados, pretende apenas decidir qual o fornecedor que melhor cumprirá o pedido para posteriormente poder invocá-lo. Assim, a lógica de negócio do sistema centra-se na eleição e consequente invocação de serviços e não na resolução dos mesmos.

Com base nestes factores é possível concluir que a fase de execução acima descrita como parte integrante de qualquer serviço, dará lugar no mediador a duas fases distintas relacionadas com a *eleição* e a *invocação* dos fornecedores de serviços.

De seguida serão apresentadas secções correspondentes às fases principais de execução presentes no mediador sendo cada uma delas explicada com maior detalhe.

4.3 Pedido

Por forma a se apresentar como alternativa à utilização de fornecedores dispersos de serviços, o sistema de mediação terá obrigatoriamente que assumir um método análogo de invocação aos seus conteúdos. Assim, esta necessidade compreende desde logo uma orientação para a utilização de um modelo baseado em *Software as a Service* onde, para um cliente, o sistema é acessível como um serviço web.

Para além de vantagens como a manutenção do sistema se encontrar apenas do lado do fornecedor, esta forma de disponibilização de software depreende que em termos práticos

não existirá qualquer diferença entre efectuar um pedido a um web service comum ou ao sistema mediador. Deste modo, torna-se necessário que o mediador possua uma forma de comunicação com o exterior semelhante a qualquer outro web service, ou seja, associando os seus conteúdos a uma `API` que especifica o que se encontra disponível para utilização e o que é necessário para o utilizar.

Um pedido a um mediador englobará o método que pretende utilizar e os parâmetros para a utilização do mesmo tal como num web service comum. O utilizador apenas terá que analisar a `API` do mediador por forma a perceber como pode efectuar o pedido e consequentemente fornecer os elementos para que tal pedido seja exequível. Do mediador, o utilizador apenas esperará a resposta. O sistema por sua vez terá que possuir um mecanismo para receber os pedidos assim como terá que disponibilizar uma `API` contendo as descrições dos tipos de serviços que suporta.

Talvez a diferença do sistema de mediação relativamente a um serviço tradicional é que este permitirá a invocação de diversos tipos de serviços, cada um com os seus argumentos, particularidades e formas de execução. No entanto, este aspecto será invisível para o utilizador já que a única diferença em termos práticos surge no conteúdo de um pedido.

Como referido acima, normalmente o utilizador constrói um pedido destinado a um fornecedor de um serviço com base numa operação e seus parâmetros, parâmetros estes que dizem respeito exclusivamente à operação em si. A diferença para a abordagem proposta surge na medida em que para além dos parâmetros que o utilizador introduz face à operação, poderá também (se assim o entender) incluir parâmetros para a eleição do fornecedor de serviço a utilizar.

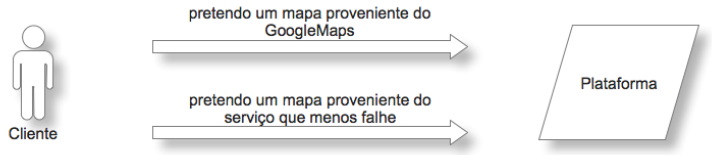

Figura 4.1: Casos de utilização da plataforma distintos

A figura 4.1 apresenta dois casos distintos de invocação de um tipo de serviço. Num deles, o utilizador requer ao mediador um serviço indicando-lhe especificamente qual o fornecedor que pretende, enquanto no outro, o utilizador indica ao mediador que pretende o fornecedor que menos falhas apresente.

Neste exemplo apresentam-se duas formas distintas de construir um pedido que consequentemente terão que possuir no mediador formas de execução distintas e que serão explicados ao longo deste capítulo.

4.3.1 Autenticação

Um aspecto relevante que pode afectar a forma como um cliente efectua pedidos ao sistema deriva dos diversos mecanismos de autenticação. Com efeito, diversos serviços pela web requerem que os seus utilizadores efectuem numa primeira instância um registo para poderem utilizar os seus conteúdos. Este registo terá então que ser utilizado pelos clientes como credenciais perante o serviço, por forma a poderem executar qualquer tipo de pedido. Por norma, estas credenciais são compostas por dois parâmetros sob a forma de identificador único perante o sistema e respectiva palavra passe.

No contexto de um sistema capaz de providenciar serviços, esta forma de invocação dos mesmos pode ser problemática. De facto, mediante o tipo de serviços a que nos refiramos, podem existir várias formas de solucionar o problema. Três soluções parecem assumir-se com maior relevância entre outras possíveis:

- o sistema conhece todos os mecanismos de autenticação do cliente e o cliente autentica-se perante a plataforma na invocação de um serviço;
- no caso de um serviço necessitar de autenticação, o cliente fornece na sequência do pedido essas mesmas credenciais, e
- o sistema possui credenciais próprias perante os fornecedores de serviços.

De facto, todas estas implementações possuem vantagens mas também desvantagens. Atente-se ao seguinte exemplo, um fornecedor de um serviço que possibilite armazenamento de fotos para o seu sistema possuirá em princípio limites relativos à capacidade máxima atribuída aos clientes. Assim é de esperar que cada cliente possua uma conta dentro do próprio serviço com esse mesmo limite.

Neste caso, seria desde logo impossível esperar que todos os clientes utilizem as credenciais do mediador. Seria no entanto possível que qualquer uma das restantes soluções fosse utilizada. No entanto, a utilização das duas primeiras soluções confere desvantagens à própria lógica de negócio do mediador. No primeiro caso é necessário para cada utilizador do mediador, que este possua registos em todos os fornecedores suportados pelo mesmo. No segundo caso, a indicação das credenciais de autenticação ao mediador estabelecem desde logo qual o fornecedor pretendido, deixando pouca margem de manobra para a resolução de falhas do serviço por parte do mediador (como por exemplo utilizar um outro fornecedor).

Deste modo, dependendo dos tipos de serviços implementados, uma das soluções propostas poderá provar-se como melhor alternativa que as restantes sendo aconselhável que

a opção seja apenas realizável no momento da implementação de um grupo de serviços no sistema.

4.4 Interpretação do pedido

A interpretação do pedido é a primeira fase de execução do mediador. Como referido anteriormente, o pedido executado pelo utilizador pode variar desde a eleição de um fornecedor específico até uma indicação de qualidade do serviço que pretenda ver executada. No entanto, antes de qualquer variação a este nível, é necessário saber qual o tipo de serviço a que se direcciona e também a operação que pretende.

Assim, torna-se necessário que a primeira fase de execução do sistema de mediação seja capaz de interpretar o pedido. No entanto, dependendo do tipo de serviço a que se direcciona, é esperado que haja diferenças numa possível execução do mediador.

4.4.1 Concepção do modelo para os serviços

Ao comparar a execução de um serviço que possua como objectivo o envio de um ficheiro com um outro serviço que execute cálculos matemáticos, é perceptível que o processo de execução dos mesmos poderá variar.

Este é um ponto importante na concepção do sistema porque permite perceber se terá que existir uma distinção entre os serviços suportados, ou seja, se cada serviço terá que ser independente dos restantes, ou se é possível conceber um modelo global capaz de agrupar todos os serviços existentes.

A resposta ao problema compreende as duas hipóteses. Na verdade, devido à diversidade de serviços na web actual é notório que existirão aspectos presentes em alguns serviços e não nos restantes. Este ponto força a uma distinção que terá que ser compreendida pelo sistema de mediação nomeadamente aquando de uma análise aos argumentos do pedido.

No entanto, existem também muitos aspectos em comum, desde a forma como os serviços são invocados até a forma como respondem. Neste sentido, é possível verificar que o modelo do sistema terá que conter zonas comuns entre os serviços, assim como zonas exclusivas a cada um deles.

Na secção anterior foi abordado que no sentido de disponibilizar aos utilizadores os serviços do sistema, estes seriam descritos numa `API` global particionada por tipos de serviços. Esta noção assume especial relevo no contexto do sistema já que as diferenças entre fornecedores de serviços concorrentes são substancialmente menores.

De facto, dois fornecedores que prestem o mesmo serviço terão argumentos necessariamente semelhantes. O mesmo sucede com as respostas, apesar de estas poderem mudar estruturalmente. Neste sentido, torna-se óbvio que fornecedores de serviços concorrentes podem de alguma forma ser agrupados.

Numa perspectiva de análise ao pedido esta distinção ganha especial relevo dado que se os conteúdos não variam, então o modo de analisar fornecedores concorrentes pode ser o mesmo.

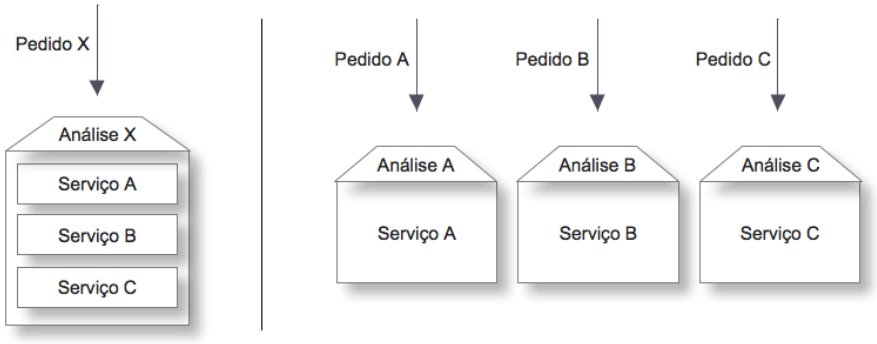

Figura 4.2: Diferenciação entre dois modelos: Um modelo com recurso ao agrupamento de serviços concorrentes face ao modelo singular

Na Figura 4.2 são apresentadas duas possíveis formas conceptuais de organização dos fornecedores de serviços onde a primeira ilustra o agrupamento dos fornecedores (segundo um tipo específico) enquanto que a segunda apresenta uma clara distinção entre todos os fornecedores de serviços.

Atentando aos requisitos do sistema, é descrita a necessidade deste ser capaz de sintetizar diversos tipos de serviços por meio de uma abstracção. Adicionalmente é ainda referido que no caso de um fornecedor não se encontrar disponível então um outro deve ser disponibilizado ao utilizador, sem que este tenha que efectuar qualquer operação. Dado isto, denote-se as seguintes vantagens relativamente à primeira implementação:

- o utilizador efectua um pedido genérico ao mediador independentemente do serviço que possa responder;
- independentemente do serviço elegido, a análise aos argumentos do pedido não precisa de variar;
- a eleição do serviço é efectuada sob a lógica dos conteúdos que estes podem fornecer (será melhor explicado na próxima secção), e
- garante independência entre tipos de serviços.

Existe no entanto uma desvantagem importante, visto que esta implementação origina uma perca de modularidade face à estrutura singular. No entanto essa mesma modularidade é obtida ao nível do tipo de serviço. Face a estes aspectos apresentados (e outros que serão abordados no capítulo da arquitectura), o modelo de agrupamento dos serviços com base em tipo de serviço acabou por ser o eleito para a construção do mediador.

4.4.2 Análise de um pedido

Depois de eleito um modo de disposição dos serviços, neste caso segundo os seus conteúdos, é necessário perceber como poderá ser efectuada uma análise aos pedidos tendo em conta este factor. Assim, uma fase de análise no contexto do mediador terá que acima de tudo perceber o que é pedido.

Um pedido, como anteriormente dito, é apenas a invocação de algo que o sistema de mediação descreve como executável na sua `API` global. No entanto, é necessário que antes de mais, o sistema seja capaz de direccionar o pedido para o tipo de serviço pretendido.

Deste modo é possível compreender que a interpretação de um pedido será composta por duas fases, uma primeira onde é analisado qual o mediador (grupo de fornecedores de serviços concorrentes) para onde deve ser reencaminhado o pedido e uma segunda onde serão analisados os conteúdos do pedido (os seus argumentos).

Esta divisão em duas fases compreende algumas vantagens a nível da implementação, nomeadamente ao deixar a cargo de cada mediador qual o serviço que melhor se presta a resolver o pedido.

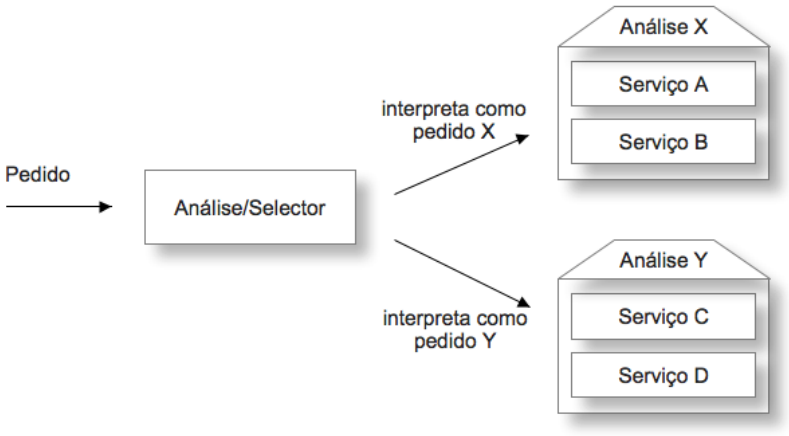

Figura 4.3: Ilustração do modo de interpretação de pedidos no sistema de mediação com base em duas fases utilizando dois mediadores

A Figura 4.3 ilustra esta divisão. Um pedido ao ser recebido pelo sistema irá em primeiro lugar ser enviado para o grupo de serviços (mediador) que melhor o possa resolver,

sendo analisado ao pormenor através da fase de análise do próprio grupo.

No entanto, a análise a um pedido pode por si só compreender diversas operações. Na secção 4.3, foram introduzidas duas formas de um cliente poder requerer algo do sistema, uma onde o cliente indica especificamente qual o fornecedor que pretende e a outra onde indica o tipo de conteúdos que quer. Estas duas formas compreendem diferentes maneiras de analisar o pedido. Se um cliente requisita um fornecedor específico ao sistema, então terá apenas de indicar qual a operação que pretende ver realizada associando para tal ao pedido os argumentos da mesma. Nestes casos, a análise apenas terá que verificar numa primeira fase, qual o grupo de serviços onde o fornecedor requisitado se encontra e posteriormente, já no contexto do mediador, verificar se todos os argumentos referentes à operação pretendida estão presentes. No entanto, se um cliente requisita um tipo de serviço, mas não indica qual o fornecedor que pretende, o mediador terá que possuir meios para decidir qual o fornecedor que lhe irá entregar uma resposta.

Neste sentido, para ajudar a esta decisão, na requisição do tipo de serviço o cliente poderá para além de indicar a operação e argumentos associados, indicar alguns parâmetros de serviço que ajudem o mediador a efectuar uma decisão. Deste modo, a fase de análise terá que ser capaz de inicialmente perceber qual o grupo de serviços a qual se direcciona o pedido e de seguida, na análise presente no próprio mediador, dissociar os parâmetros de modo a que estes possam ser utilizados em diferentes alturas da execução.

A forma de execução referente a este segundo caso resolve também um outro problema associado à criação de um middleware para serviços capaz de oferecer a noção de concorrência. No contexto de alguns serviços, existem operações onde é impossível oferecer ao utilizador um fornecedor concorrente. Por exemplo, no caso de um serviço de fotos, se o utilizador pretender obter uma foto alojada num determinado fornecedor, é impossível ao sistema providenciar outro fornecedor que lhe possa fornecer essa mesma foto.

Nestes casos, a plataforma não pode aplicar o modelo de abstracção. Como tal, o cliente não poderá requisitar ao sistema um serviço genérico capaz de lhe fornecer a foto. No caso de o serviço de fotos se encontrar offline, então o mediador pouco pode fazer para evitar o não cumprimento da operação. O máximo que este pode fazer é disponibilizar face a esses serviços as operações passíveis de concorrência obrigando o utilizador no caso de a operação ser exclusiva a um fornecedor, a indicar directamente o fornecedor de serviço que pretende.

Assim, o segundo caso acima apresentado adquire maior importância no sistema, permitindo que o utilizador possa conceber a sua aplicação utilizando apenas o sistema de mediação mesmo que este tenha a necessidade de utilizar fornecedores de serviços especí-

ficos. Deste modo, o utilizador não necessita de perceber duas `APIs`, simplesmente utiliza o sistema e no caso de existirem operações não concorrentes então apenas necessita de indicar o fornecedor desejado.

4.5 Eleição do melhor fornecedor de serviço

Um dos requisitos mais importantes enunciados anteriormente centra-se na capacidade do sistema em fornecer ao cliente o melhor serviço disponível de entre um grupo de fornecedores de serviços elegíveis. Para tal, é necessário que um mediador possua mecanismos que lhe permitam optar por um fornecedor em detrimento de outro, assim como possuir bases nas quais fundamentar as suas escolhas.

Como tal, e tendo em conta o modo de funcionamento de um serviço através da web, será lógico que características como a velocidade de resposta ou latência média sejam analisados pelo sistema de mediação e consequentemente introduzidos nos mecanismos de escolha. Para tal, é necessário que o sistema seja capaz de analisar esses mesmos valores com base numa monitorização que deve ser constante por força da instabilidade das comunicações em rede.

4.5.1 Monitorização

Dado ser sobre os serviços que maioritariamente recai a lógica de negócio apresentada, torna-se essencial a presença de um elemento de monitorização capaz de recolher estatísticas sobre os fornecedores desses mesmos serviços. No caso de se tratar de uma arquitectura que incorpore serviços remotos, este elemento torna-se mesmo obrigatório para administradores do sistema poderem obter *feedback* relativo ao funcionamento do mesmo, e perceber se os serviços incorporados apresentam uma qualidade aceitável. Com base nesta análise, os administradores podem, no caso de entenderem necessário, criar meios por forma a melhorar o desempenho do serviço ou até substituí-lo no sistema por um outro equivalente, procurando melhorar o desempenho global do sistema de mediação.

No caso particular em estudo, a monitorização possui um objectivo adicional apresentado já por várias vezes, focalizado em dotar o sistema de meios para que um futuro cliente possa eleger um fornecedor de serviço com base em aspectos que, no seu entender, possam ser mais relevantes. De facto, dado tratar-se de uma arquitectura orientada aos serviços (possivelmente remotos), para além do tempo de execução do serviço em si, outros factores poderão afectar a entrega dos conteúdos atrasando assim o processo de execução do cliente. Nestes casos, perceber se um fornecedor de serviço é capaz de en-

tregar uma resposta num tempo menor que um fornecedor concorrente pode ser decisivo para o cliente na questão de eleger um deles.

Assim, o sistema desenvolvido terá que analisar aspectos derivados da comunicação entre os fornecedores de serviços e o sistema de mediação, assim como obter tempos relativos de execução em função de determinadas operações. Para tal, é necessário que o sistema possua um módulo próprio contendo testes equitativos aos diferentes fornecedores e seja capaz de interpretar esses resultados. De seguida são apresentados alguns desses aspectos, normalmente designados por aspectos de monitorização passiva, assim como a sua utilidade neste contexto:

- *Tempo de resposta* - Mede o tempo total decorrido entre o efectuar do pedido e a obtenção da resposta ao mesmo. Estes valores podem indicar a capacidade do servidor em proporcionar aos seus clientes tempo constantes de execução mesmo em alturas de um maior pico de utilizadores.

- *Latência* - Mede o tempo decorrido entre o efectuar do pedido e a recepção do mesmo por parte do fornecedor de serviço. Como tal, este aspecto pode ser crítico no caso de ser necessário transmitir um grande volume de dados.

- *Throughput* - Permite calcular o número de elementos processados por unidade de tempo. Por forma a ser comparável, neste caso o cálculo será obtido através da razão entre o número total de pedidos e o intervalo de tempo definido.

- *Largura de Banda* - Permite calcular possíveis constrangimentos nas comunicações entre o fornecedor e o sistema de mediação durante a execução do pedido. Neste caso, a sua obtenção é baseada no cálculo da transferência de dados máxima que uma ligação pode suportar num dado intervalo de tempo.

- *Status* - Verifica se um serviço se encontra online. Permite posteriormente analisar o número de falhas de um fornecedor num determinado intervalo de tempo.

No entanto, estes valores são valores obtidos num dado momento e podem não ser representativos do normal funcionamento dos serviços. Como tal, devem ser analisados e dispostos ao cliente através de um cálculo médio com base em métricas. No contexto deste sistema fazem sentido métricas que permitam obter aspectos tais como:

- *Disponibilidade* - Devolve a percentagem de disponibilidade de um serviço. Calcula a capacidade de um sistema se encontrar acessível com base numa razão entre o número total de monitorizações efectuadas e o número de respostas positivas obtido.

- *Desempenho* - Calcula um valor médio com base nos resultados obtidos como tempo de resposta. Por norma no cálculo deste métrica, o pior valor obtido é excluído do cálculo. Permite perceber qual o serviço que mais rapidamente (em média) responde a um pedido.

- *Tempos Médios* - Compõe valores médios relativos com base nos restantes valores acima descritos (throughtput, latência ou largura de banda). Para além de proporcionar mais elementos de selecção para o cliente, permite simultaneamente que os administradores do sistema possam analisar o estado das comunicações com os fornecedores dos serviços.

Estes cálculos por sua vez são efectuados com base num determinado intervalo de tempo que será definido pelo administrador do sistema. Ao cliente, deve ser proporcionada na `API` do sistema a introdução dos parâmetros que pretende ver incorporados no cálculo do fornecedor de serviço. Deste modo e de acordo com o seu pedido, é lhe garantido que o fornecedor que lhe responde é o melhor de uma gama de fornecedores elegíveis.

Adicionalmente, o cálculo destes parâmetros deve ser contínuo dada a possível alteração de condições ou circunstâncias do serviço tendo que acontecer o mesmo com as métricas. Para tal, é necessário que o sistema possua condições para que a monitorização possa ocorrer continuamente.

4.5.2 Funcionalidades de serviço

Não obstante a importância da monitorização, a eleição de um fornecedor de serviço pode compreender aspectos talvez mais relevantes para o utilizador do que características de comunicação. Apesar de existir uma abstracção ao nível do pedido que permite que um utilizador receba um conteúdo sem ter qualquer informação de qual o fornecedor que o executou, podem existir casos em que um ou dois fornecedores presentes num grupo possuam uma funcionalidade extra desejada pelo utilizador.

Por exemplo, um grupo de serviços de mapas contêm dois fornecedores que permitem ao utilizador obter na sua implementação do mapa uma visualização em três dimensões. Neste caso, será útil que o cliente possa referir que pretende um serviço com essa mesma funcionalidade.

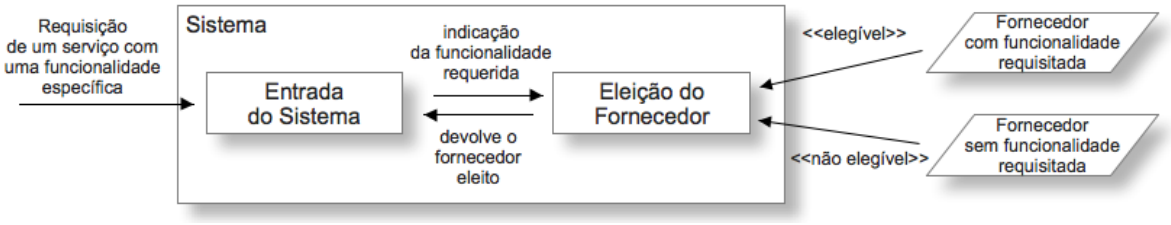

Figura 4.4: Exemplificação do processo de eleição de um fornecedor com base em funcionalidades indicadas na invocação do sistema

Ilustrada na Figura 4.4 encontra-se um exemplo desta implementação. Como demonstrado, a indicação da funcionalidade pode ser incluída no momento de invocação do pedido, levando a que na altura de eleição de um fornecedor o sistema dê preferência aos fornecedores de serviços que a possuem. Para que tal seja possível, é necessário no entanto que no momento de desenvolvimento do sistema de mediação, nomeadamente ao nível da constituição do mediador correspondente, seja efectuada uma análise às capacidades dos fornecedores através de um levantamento de quais as características que podem ser pertinentes para um futuro cliente. Estas características por sua vez terão de ser armazenadas, por forma a poderem ser consultadas pelo mediador no momento de execução.

Por forma a disponibilizar estes aspectos aos clientes, será necessário que a `API` do mediador reflicta a existência destas funcionalidades, permitindo ao utilizador que as refira no momento de invocação de um pedido. Adicionalmente, poderá ser necessária ainda a elaboração de uma abstracção no momento da resposta dada a possível inclusão de conteúdos não presentes em todos os fornecedores.

Outro aspecto presente na constituição do módulo de eleição de serviço, provém da diferença entre funcionalidades existentes em determinados tipos de serviços face aos restantes. É necessário que, tal como acontece com a fase de análise anteriormente apresentada, o módulo de eleição de serviços seja orientado a um grupo de fornecedores em particular. Desta forma é assegurado que a eleição de um fornecedores recaí apenas sobre fornecedores concorrentes entre si. Adicionalmente, no caso de um dos fornecedores do serviço se encontrar indisponível, a atribuição de um fornecedor concorrente capaz de fornecer as mesmas funcionalidades é facilitada, dada a restrição entre fornecedores do mesmo grupo.

Em conclusão, o processo de eleição do fornecedor de serviço compreende essencialmente a conjugação entre os resultados de monitorização, as funcionalidades de serviço e o pedido executado pelo utilizador. Será necessário que cada mediador possua um módulo capaz de interpretar as características envolvidas no pedido de acordo com os dados da monitorização que possui, para que consequentemente de acordo com os fornecedores de

serviço que se encontrarem disponíveis, ser capaz de eleger o fornecedor a ser invocado.

4.6 Invocação do serviço

Na base do sistema a desenvolver encontram-se os fornecedores de serviços. No sistema de mediação, serão estes fornecedores em primeira instância, os responsáveis pela criação de uma resposta para o cliente. Para que isto aconteça, é necessário que o sistema seja capaz de os invocar de forma correcta com base no pedido efectuado pelo cliente.

Por norma, um fornecedor de serviço disponibiliza os seus conteúdos através de operações específicas para o efeito. De modo a dar a conhecer estas operações, o serviço é descrito numa `API`, onde este específica a forma de invocação do mesmo. Será através desta `API`, que os clientes poderão perceber de que forma podem efectuar um pedido ao fornecedor, quais os argumentos que este espera e mesmo saber qual será o formato da resposta.

Por forma a incluir no sistema diversos fornecedores, é necessário desde logo que a cada fornecedor corresponda um módulo específico. Isto acontece devido às possíveis diferenças entre as `API`s dos fornecedores contidos no mediador e seus modos de invocação. De facto, pode até mesmo acontecer que um fornecedor possua uma operação não suportada por um outro. Através da separação de fornecedores por módulos, este problema não se coloca sendo desde logo garantido que possíveis diferenças entre processos de invocação a fornecedores de serviços e consequentes formatos de resposta não requerem uma limitação entre as operações suportadas pelo mediador.

Na constituição de cada módulo, é necessário que em primeiro lugar o implementador analise a `API` do respectivo fornecedor. De seguida, e com base nesta análise, terá que representar na estrutura do módulo todas as funcionalidades que o serviço possua mapeando estas mesmas funcionalidades em operações que possam posteriormente ser requeridas na execução do mediador. Esta modularidade patente nos serviços, permite simplificar a futura adição de fornecedores à plataforma. Assim, e por forma a introduzir um novo fornecedor na plataforma, será apenas necessária a constituição de um novo módulo representativo dos métodos de invocação da `API` do serviço em causa.

Com isto, pretende-se que na sua execução o mediador tenha simplesmente que invocar o serviço pré-eleito, associando-lhe apenas os parâmetros de serviço contidos no pedido. Caberá ao módulo constituir o formato correcto de invocação do fornecedor, proceder à sua invocação e posteriormente captar os resultados por forma a estes poderem no futuro ser transmitidos ao cliente.

4.7 Devolução dos resultados

Apesar da modularidade presente na invocação do serviço possuir inúmeras vantagens, esta constituí também um problema na forma como os resultados serão entregues ao cliente. De facto, poderão existir situações em que o formato de resposta de um serviço não é linear, ou seja, poderão existir casos em que o conteúdo fundamental da resposta vem contido numa estrutura própria do fornecedor.

Nestes casos, é levantado um problema para o cliente. Se de facto é pretendido que um cliente possa invocar um tipo de serviço à plataforma e lhe seja atribuído um fornecedor em particular, o cliente não estará à espera de perceber qual o fornecedor atribuído, assim como não estará à espera que consoante o fornecedor, lhe sejam entregues os resultados de diferentes formas.

Como tal, caberá ao mediador sintetizar os diferentes resultados provenientes dos diversos fornecedores numa única forma. Com isto, pretende-se que o cliente final apenas necessite de conhecer um formato de resposta preparando assim a sua aplicação para interpretar os resultados contidos neste.

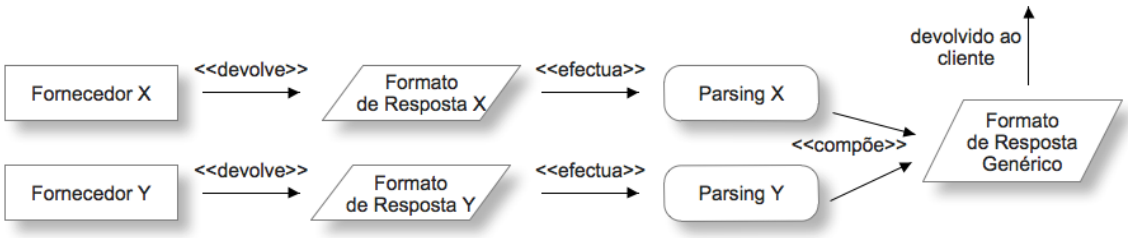

Figura 4.5: Processo de Uniformização de Resultados Provenientes de Diversos Fornecedores

Exemplificado na Figura 4.5 encontra-se o processo de uniformização dos resultados num único formato de resposta. Por forma a conceber este formato o implementador terá que analisar os diferentes resultados provenientes dos diversos fornecedores, procurando levantar todos os resultados possíveis dos mesmos. De seguida, irá proceder a uma comparação entre os mesmos por forma a tentar conceber uma estrutura global capaz de conter os resultados provenientes dos diferentes fornecedores. De facto, dependendo do tipo de resultados possíveis devido à presença de operações distintas nos serviços, várias estruturas poderão mesmo ter que ser concebidas. No entanto, apesar da morosidade deste passo, por norma ele será facilmente resolúvel devido ao facto de se tratarem de fornecedores de serviço concorrentes onde existirão provavelmente semelhanças entre as diferentes respostas.

Após a sua concepção, o implementador terá que constituir no mediador métodos de

"parsing" orientados a cada fornecedor de forma a que seja constituído um mapeamento entre os resultados dos serviços e a estrutura global definida. Esta estrutura será assim a única estrutura que o cliente irá receber.

Por fim, quando a invocação do pedido e consequente (se necessária) abstracção dos resultados estiver concluída, estes serão então devolvidos ao cliente através do mesmo canal de comunicação utilizado para receber o pedido.

No próximo capítulo será constituído um modelo arquitectural capaz de sintetizar os requerimentos descritos ao longo deste capítulo, de forma a que o sistema de mediação pretendido possa ser construído.

Capítulo 5

Modelo Arquitectural para o Sistema de Mediação

O modelo arquitectural definido neste capítulo baseia-se nos principais conceitos discutidos na fase de concepção apresentada anteriormente. Este modelo pretende assumir-se perante os clientes como um multi-mediador (inspirado no modelo de mediação presente em SOA) em que cada mediador de serviço implementado será capaz de proporcionar ao cliente o melhor fornecedor de serviço correspondente ao tipo de conteúdo solicitado. Para tal, cada fornecedor de serviço será monitorizado pelo sistema de forma a ser executada uma análise rigorosa relativa ao seu funcionamento. Através desta análise, pretende-se que seja possível efectuar uma avaliação do fornecedor que permita ao sistema eleger aquele no qual possa delegar os pedidos.

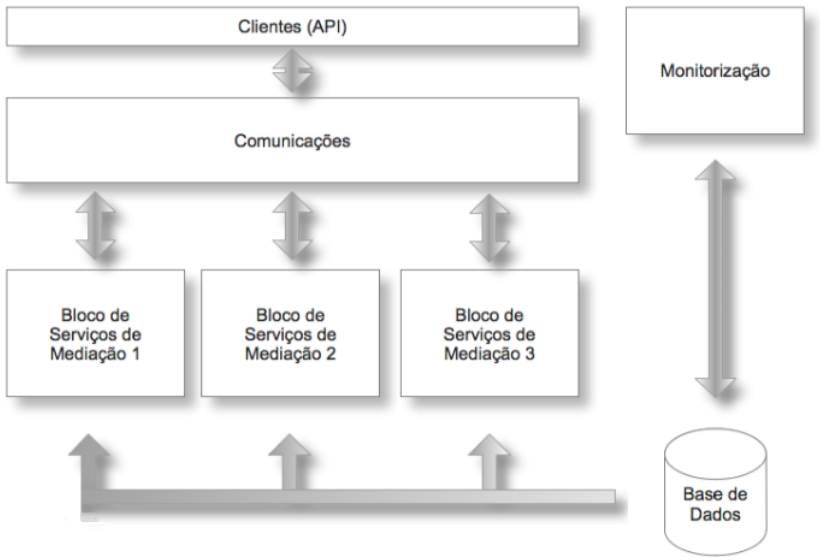

Figura 5.1: Visão global do modelo arquitectural proposto para o sistema de mediação.

A Figura 5.1 apresenta a disposição global do modelo arquitectural. Na sua constituição, o modelo é composto fundamentalmente por três áreas: uma área relativa às comunicações, uma área relativa aos mediadores de serviço e por fim uma área de monitorização. Por forma a dar suporte às operações de monitorização, o sistema possuirá ainda uma base de dados própria.

Na perspectiva do cliente, o sistema será acessível através de uma `API` concebida com base nas operações presentes em cada tipo de serviço incorporado. A forma de disponibilização de conteúdos assenta sobre o conceito de `SaaS`, que implica que os conteúdos sejam acedidos remotamente (por meio de Web Services), mas com a execução do sistema a correr do lado do fabricante.

Resumidamente, o processo de execução desta arquitectura assenta essencialmente em duas fases. A primeira fase inicia-se quando um pedido é recebido pelo bloco de comunicações. Este bloco é responsável por perceber se o pedido se encontra bem formado, se pode ser executado e qual o bloco de serviços de mediação que terá de o executar. Se estes aspectos forem concretizados, o bloco de comunicações dará então início à segunda fase.

A segunda fase inicia-se quando um pedido é redireccionado para um dos blocos de serviços de mediação. Cada um destes blocos será elaborado com base em fornecedores de serviços concorrentes agrupados numa perspectiva de tipo de serviço. Assim, cada bloco de serviços de mediação terá como finalidade analisar detalhadamente o pedido por forma a eleger qual o fornecedor que melhor poderá prestar o serviço e consequentemente invocá-lo por forma a posteriormente compor um resultado.

Apesar do bloco de monitorização não ser um elemento activo na execução da plataforma, é em parte através da sua acção que os fornecedores são eleitos. Neste caso, a monitorização funciona de uma forma cíclica, baseando-se em aspectos relacionados com as ligações aos serviços envolvidos na plataforma. Estes aspectos serão recolhidos pela monitorização e consequentemente introduzidos na base de dados do sistema. A intenção é que seja possível, por exemplo, que a plataforma eleja um fornecedor de serviço simplesmente porque é o que mais rapidamente irá processar a resposta, baseando-se para tal nos valores anteriormente recolhidos.

Nas próximas secções ir-se-á explorar detalhadamente cada um destes blocos e seus constituintes.

5.1 Bloco de Comunicações

O primeiro bloco da plataforma designa-se por comunicações. Este bloco, tal como o nome indica, é responsável por todas as comunicações entre os clientes e a plataforma. A sua constituição é elaborada essencialmente a partir da `API` do sistema construído. Qualquer pedido enviado ao sistema de mediação será recebido por este bloco, que por sua vez o interpretará e encarregar-se-á de o redireccionar para o bloco de serviços de mediação apropriado.

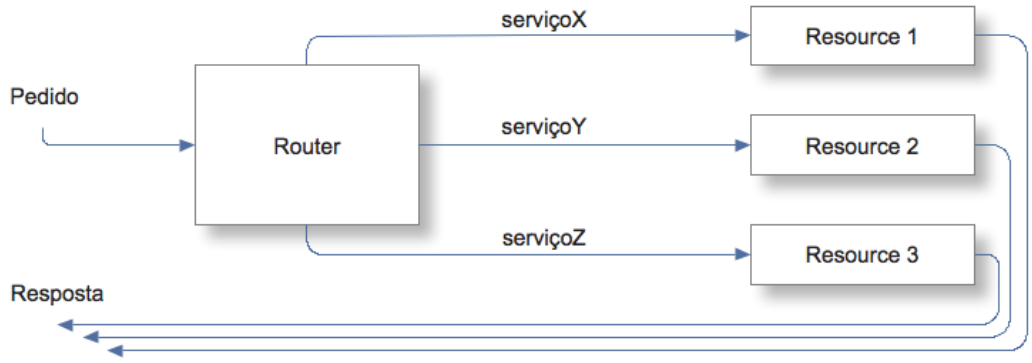

Figura 5.2: Constituição interna do bloco de comunicações.

O modo de operação deste bloco, assenta sobre dois tipos de módulos (ver Figura 5.2). O primeiro, designado por *router*, recebe o pedido e, com base na operação recebida, invoca o módulo *resource* capaz de processar o pedido. Cada um destes módulos *resource* será orientado a um tipo de serviço em particular tendo a função de interpretar o pedido recebido. Esta interpretação do pedido, com base na operação (presente na `API`) invocada e correspondentes parâmetros, analisará deste logo se o pedido é exequível pela plataforma, sendo que no caso de não o ser o pedido será desde logo cancelado sem que nada chegue ao bloco de serviços de mediação.

Para além do tratamento do pedido, este bloco é também o responsável pela entrega da resposta. Esta, ou é gerada localmente, no caso já referido de pedido não válido, ou é recebida do bloco de serviços de mediação e reencaminhada para o cliente sem qualquer alteração.

Independentemente do número de mediações que a arquitectura implemente, o bloco de comunicações é único. Haverá, obviamente, vários blocos *resource*, mas, como se verá adiante, não necessariamente um por bloco de serviços de mediação.

5.1.1 *Router*

O primeiro módulo do bloco de comunicações tem de facto o papel de um *router*. A sua função prende-se com a recepção de um pedido, sua análise e consequente redireccionamento para o módulo *resource* apropriado.

Como tal, por cada pedido que chegue à plataforma, o *router* encarregar-se-á de o passar a um dos *resources* disponíveis com base na operação pretendida pelo cliente. A escolha do *resource* será efectuada com base numa lista de correspondência contida no próprio *router* entre as operações suportadas pela plataforma e os *resources* disponíveis.

Por exemplo: imagine-se um pedido efectuado à plataforma para a operação "criarMapa". O *router*, ao receber este pedido, e com base nos *resources* que conhece, analisará se algum deles possui como função responder a pedidos direccionados a "criarMapa". Se assim acontecer, então invocará esse *resource* delegando-lhe o pedido, incluindo todos os seus argumentos, terminando a sua função.

De salientar que por cada pedido que chegue à plataforma, o *router* irá invocar apenas um *resource*.

5.1.2 *Resources*

Se um pedido recebido pelo *router* corresponde a uma operação suportada pelo sistema de mediação, um módulo *resource* é solicitado a continuar com o seu processamento. A primeira função deste módulo é a de avaliar se o pedido se encontra bem formado, ou seja, se contém os argumentos obrigatórios conforme especificado na descrição de serviços da plataforma (`API`).

Cada módulo *resource* está associado a um bloco de serviços de mediação. Se um pedido recebido por um módulo *resource* está bem formado, este encaminha-o para o bloco de serviços de mediação que lhe está associado, ficando a aguardar a resposta, para posteriormente a entregar ao cliente. Se o pedido não está bem formado, o próprio *resource* produz uma resposta de rejeição. Este procedimento garante que a camada de serviços de mediação apenas é solicitada se de facto for necessário, aliviando assim a sua carga de execução.

Cada *resource* está associado a um e um só bloco de serviços de mediação. Mas, pode lidar com várias operações sobre esse serviço. Desta forma, e independentemente da implementação da componente de comunicações (`REST`, `SOAP` ou outro), cada *resource* terá um número de operações que suportará. Por exemplo, e supondo um caso `REST`, um *resource* poderá responder a pedidos "Get" ou "Post" se estes se encontrarem definidos, sendo que nenhum deles é obrigatório.

Não há uma correlação directa entre o número de *resources* e o número de blocos de serviços de mediação. Como dito anteriormente, um *resource* está associado a apenas um bloco de serviços de mediação. Mas, nada impede que *resources* diferentes estejam conectados ao mesmo bloco. Este facto é ilustrado na Figura 5.3, onde os *resources* 3 e 4 estão conectados ao mesmo bloco de serviços de mediação, o mais à direita. Assim, apesar da existência de um número máximo de operações por *resource*, o número de operações sobre uma dado serviço de mediação é ilimitado.

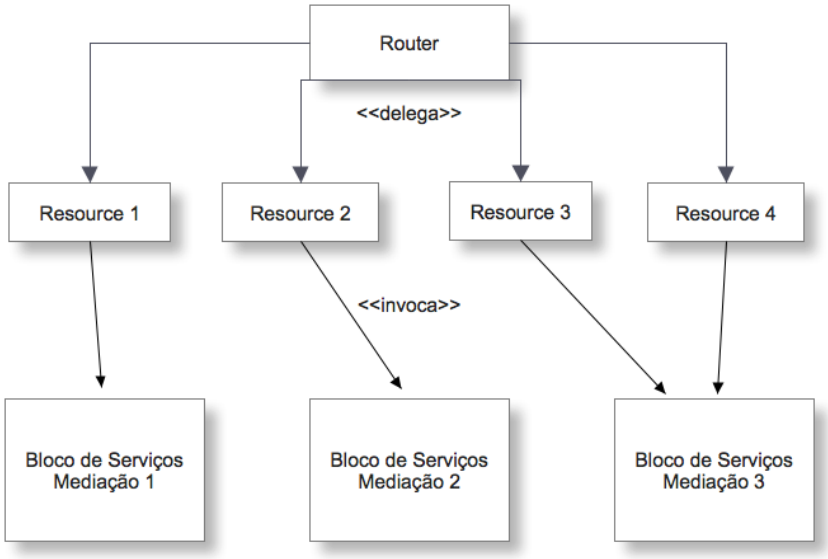

Figura 5.3: Modelo de execução entre *resources* e serviços de mediação

Os *resources* são independentes entre si, mesmo que lidem com o mesmo bloco de serviços de mediação. Este facto, além de dotar o sistema de uma boa modularidade, acautela a expansibilidade do sistema, em termos do incremento das operações disponibilizadas. Ao nível do bloco de comunicações a incorporação de novas operações corresponde ao desenvolvimento de novos *resources* e à actualização da lista de correspondências no *router*.

5.2 Bloco de Serviços de Mediação

O bloco de serviços de mediação é a peça central do modelo arquitectural proposto. De facto, ele implementa uma área de negócio do sistema de mediação. No modelo arquitectural proposto haverá tantos blocos deste tipo quantos os serviços de mediação que o sistema implementa.

São claras as vantagens desta abordagem. O modelo arquitectural proposto, tal como

já foi referido anteriormente, permite a incorporação no mesmo sistema de mais do que um serviço de mediação. Por exemplo, poder-se-á construir um sistema que sirva de mediador na obtenção de mapas e de mediador na obtenção de música. O processamento dos pedidos dos vários tipos de mediação serão seguramente diferentes, assim como serão diferentes os tipos de resultados devolvidos. Isto aconselha ao desenvolvimento separado de cada uma das mediações. Mais, as mediações diferentes poderão mesmo ser desenvolvidas por equipas de desenvolvimento diferentes. A modularidade resultante da separação em blocos diferentes por mediação favorece claramente o desenvolvimento.

A capacidade de expansão do sistema em termos do número de mediações está também assegurada, embora a inclusão de novas mediações corresponde, a este nível, à construção de novos blocos de serviços de mediação.

Cada bloco de serviços de mediação tem sobre a sua alçada todos os fornecedores de um dado tipo de serviço. Embora prestando serviços equivalentes, cada fornecedor terá a sua própria `API` de interacção, quer em termos da forma como o pedido deve ser feito, quer no formato de apresentação dos resultados. Compete a cada bloco de serviços de mediação definir uma camada de abstracção que permita aos clientes interagirem com todos os fornecedores da mesma maneira.

Na realidade, frequentemente, o cliente nem saberá com que fornecedor estará a interagir, porque este será escolhido pelo mediador com base em parâmetros fornecidos pelo cliente aquando do seu pedido. Cada bloco de serviços de mediação, com base em informações sobre os fornecedores de serviços em questão, será capaz de escolher aquele que melhor pode preencher os requisitos do utilizador. Parte dessas informações resultarão de um trabalho de recolha e tratamento de dados realizada pelo bloco de monitorização (ver Figura 5.1), de que se falará mais adiante.

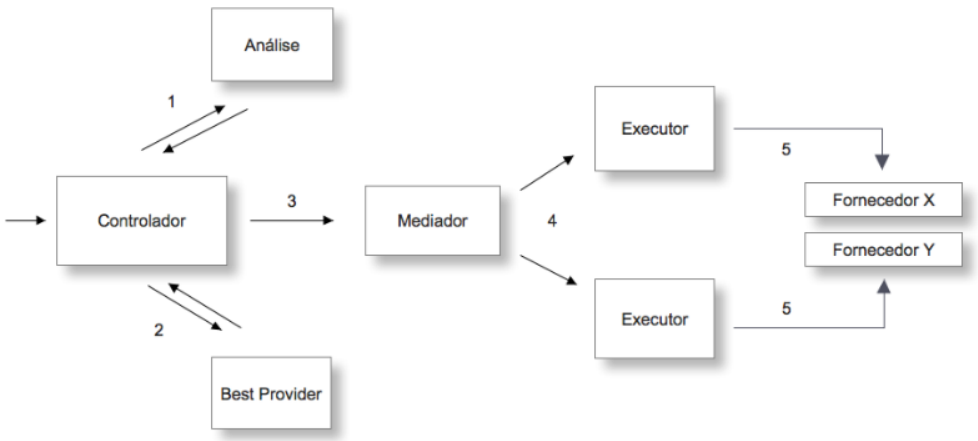

Figura 5.4: Visão global do bloco de serviços de mediação.

Na Figura 5.4 apresenta-se um diagrama com os blocos constituintes de um bloco de serviços de mediação, considerando dois fornecedores de serviços. Toda a actividade é comandada pelo bloco *controlador*. As etiquetas nos arcos mostram a ordem pela qual os outros blocos participam no processamento de um pedido. Por ordem e resumidamente, os seus papeis são os de analisar e estruturar o pedido, escolher o melhor fornecedor e, finalmente, realizar o pedido de forma a compor a resposta a enviar ao cliente.

5.2.1 Analisador

No bloco de comunicações faz-se uma primeira análise aos pedidos que chegam ao sistema de mediação. Esta análise preocupa-se apenas com a verificação da existência dos parâmetros obrigatórios de um pedido. Mas, há outros parâmetros que podem estar presentes. O bloco Analisador é a componente do bloco de serviços de mediação responsável pela análise conjunta de todos os parâmetros e pela sua estruturação de forma a serem usados pelos restantes blocos.

Os parâmetros presentes num pedido podem ser agrupados em duas categorias de acordo com a sua natureza: *parâmetros de serviço* e *parâmetros de eleição do fornecedor*. Os primeiros podem ser definidos como todos aqueles que dizem respeito ao método invocado e que serão utilizados para resolver o pedido. Por exemplo, no contexto de criação de um mapa, um parâmetro de serviço poderá ser a indicação das coordenadas geográficas onde o mapa se deverá centrar. Há parâmetros de serviço obrigatórios e opcionais. A existência dos primeiros é assegurada no bloco de comunicações.

Os parâmetros de eleição de fornecedor são usados para escolher o fornecedor do serviço que melhor satisfaz os requisitos solicitados pelo cliente. Devem ser divididos em duas sub-categorias. Por um lado, há parâmetros que se referem a variáveis estatísticas gerais sobre o desempenho dos fornecedores, independentemente da sua função, e que resultam de uma monitorização activa e continuada (ver secção 5.3). Como exemplos deste tipo de parâmetros pode-se referir a disponibilidade, a latência e o tempo de resposta. Por outro lado, há parâmetros que se referem a características específicas de um serviço. Por exemplo, num serviço de armazenamento temporário de ficheiros, a durabilidade do armazenamento pode ser um factor importante na escolha do fornecedor do serviço.

O papel do bloco Analisador é o de extrair os parâmetros do pedido, verificar a sua correcção sintáctica e semântica e agrupá-los em registos de acordo com a sua natureza. Daqui resultarão dois registos, um com os parâmetros de serviços e outro com os parâmetros de eleição de fornecedor. Este último, por sua vez, está dividido em parâmetros gerais e específicos.

O registo com os parâmetros de eleição de fornecedor será enviado ao bloco *Best Provider* para permitir a escolha do fornecedor. O registo de parâmetros de serviço será enviado ao bloco Mediador, conjuntamente com a indicação do fornecedor escolhido, para se proceder à prestação propriamente dita do serviço. Este procedimento está ilustrado na figura 5.5.

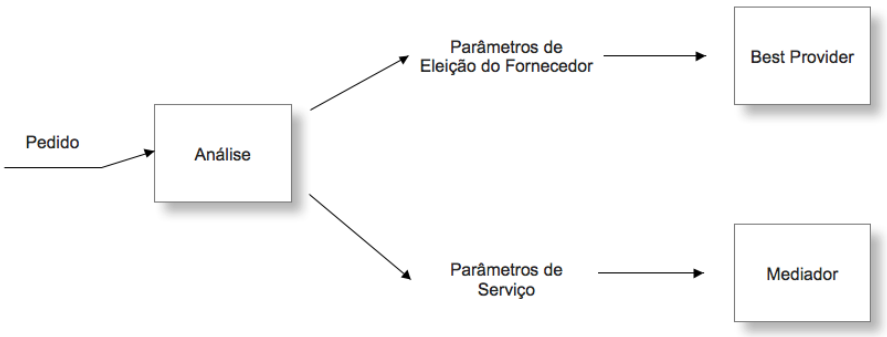

Figura 5.5: Separação de parâmetros efectuado pelo bloco de análise

5.2.2 Controlador

O bloco Controlador (ver Figura 5.4) funciona como elemento de interface entre o bloco de comunicações e o bloco de serviços de mediação. É responsável por receber um pedido de serviço, desencadear o conjunto de procedimentos internos necessários à sua execução e devolver um resultado ao bloco de comunicações.

Estruturalmente o controlador será composto por um conjunto de métodos distintos que poderão ser invocados pela camada de comunicações. Estes métodos têm uma correspondência directa com as operações suportadas pelo tipo de serviço que o bloco de serviços de mediação presta. Note-se que nem todas as operações têm de ser suportadas por todos os fornecedores associados a esta mediação. Na realidade, cada método representa uma operação que pelo menos um fornecedor de serviço é capaz de prestar.

Olhando o modelo interno do bloco de comunicações (ver Figura 5.3) verifica-se que o Controlador pode ser acedido por diferentes *resources*. No entanto, isso não influencia a construção deste, e como tal na sua definição seja irrelevante que haja um ou mais *resources*. Ele configura-se apenas como um conjunto planar de métodos, um por cada operação disponibilizada.

Normalmente, o modo de funcionamento do Controlador é composto por três fases, que se sucedem. Este processamento é análogo para todas as operações presentes num bloco de serviços, independentemente do tipo de serviços prestado.

Após receber o pedido do bloco de comunicações o Controlador passa-o ao Analisador para que este avalie sintáctica e semanticamente os parâmetros e os estruture em dois conjuntos, os parâmetros de serviço e os parâmetros de eleição de fornecedor. Na segunda fase, o Controlador invoca o Best Provider, passando-lhe a operação pretendida e o conjunto dos parâmetros de eleição do fornecedor, para que este escolha o fornecedor do serviço. Finalmente, na terceira e última fase, o Controlador pede ao Mediador para executar o pedido, passando-lhe a operação pretendida, o fornecedor escolhido e o conjunto de parâmetros de serviço. A resposta fornecida pelo Mediador é encaminhada para o bloco de comunicações.

O processamento pode não chegar à terceira fase. Podem surgir erros quer no Analisador quer no Best Provider que forcem o fim do processamento e o consequente envio de uma resposta de erro apropriada.

5.2.3 *Best Provider*

Um dos objectivos do modelo arquitectural proposto é a possibilidade do cliente condicionar a escolha do fornecedor de serviço. Na realização do pedido, o cliente é convidado a introduzir (com base na `API`) alguns parâmetros que influenciam a eleição do fornecedor. Estes parâmetros são extraídos do pedido pelo bloco Analisador (ver secção 5.2.1) e são fornecidos ao bloco *Best Provider* para que este, com base neles, escolha o fornecedor que melhor pode prestar o serviço, de acordo com as condições determinadas pelo cliente.

Tal como foi dito na secção 5.2.1, há dois tipos de parâmetros, gerais e específicos. Os parâmetros gerais são independentes do tipo de serviço e estão relacionados com o desempenho médio dos fornecedores. Representam atributos tais como disponibilidade, latência ou tempo de resposta. Uma unidade de monitorização, parte integrante do sistema de mediação e de que falará mais adiante, está periodicamente a avaliar os fornecedores e a determinar esses atributos, colocando-os numa base de dados para que o *Best Provider* os possa consultar.

Os parâmetros específicos referem-se a particularidades de um dado serviço, que podem fazer com que um dado fornecedor seja seleccionado ou preterido. Por exemplo, um fornecedor de serviços de meteorologia pode ser preterido porque não oferece informação sobre pressão atmosférica ou porque não cobre a região pretendida. Os atributos desta classe representam limitações ou especificidades dos fornecedores de serviços cobertos por uma dada mediação.

O processamento no *Best Provider* está escalonado em duas partes. Primeiro são analisados os parâmetros específicos, seguindo-se a avaliação dos parâmetros globais. Esta

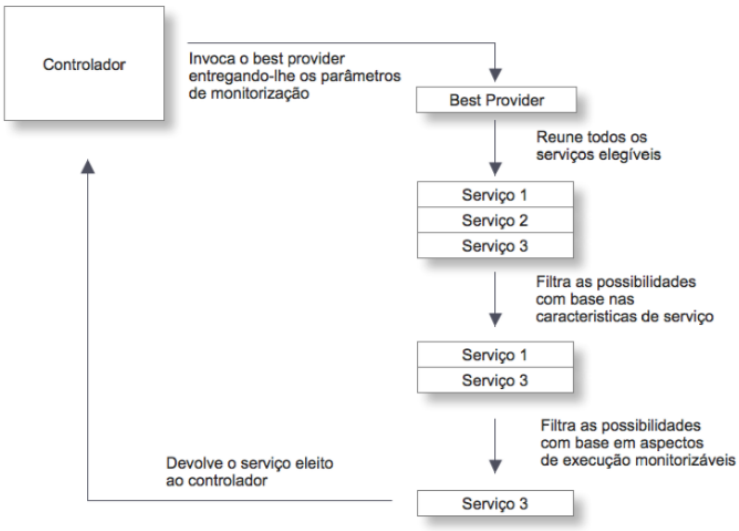

Figura 5.6: Sequência de execução no processo de eleição do melhor fornecedor de serviço.

ordem assenta sobre o princípio lógico de que não faz sentido recomendar um determinado fornecedor de serviço se este não possuir uma característica desejada pelo cliente. A Figura 5.6 ilustra esse processamento. O *Best Provider* começa por reunir uma lista de todos os fornecedores activos que preencham as características de serviço desejadas pelo utilizador. De seguida, com base em características globais, como disponibilidade ou desempenho, o módulo seleccionará uma das entradas na lista devolvendo de seguida o fornecedor eleito para o controlador.

Pode acontecer que, depois do processo anterior, dois ou mais fornecedores sejam eleitos. Fica ao critério do implementador decidir o critério de desempate. Pode ainda acontecer que nenhum fornecedor cumpra os requisitos solicitados pelo cliente. Neste caso o sistema reportará essa informação ao cliente.

5.2.4 Mediador

O mediador é um bloco simples que funciona essencialmente como um elemento de comutação que encaminha a operação invocada pelo controlador para um dos blocos de execução que tem debaixo da sua alçada (ver Figura 5.1). Desempenha um papel aparentemente menor, mas, na realidade, contribui para uma clara separação de conceitos.

Quando invocado, o mediador recebe do controlador a operação pretendida, a identidade do fornecedor que deve ser contactado e a lista de parâmetros de serviço e, em consequência, encaminha a operação e os parâmetros para o bloco de execução associado ao fornecedor especificado e fica a aguardar a resposta que, quando chega, se limita a encaminhar para o controlador.

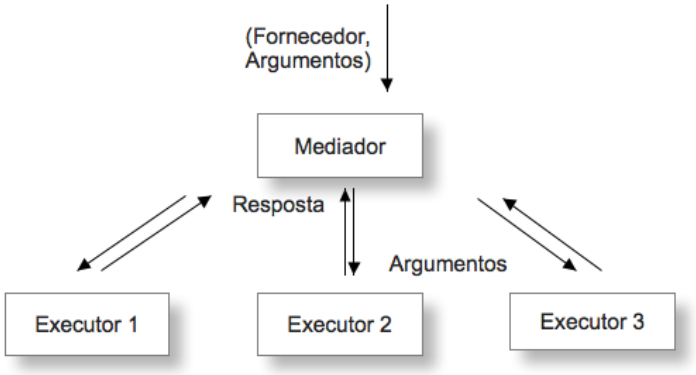

Figura 5.7: Modelo de funcionamento do mediador

5.2.5 Executor

Os executores são os componentes do bloco de mediação responsáveis pelo contacto directo com os fornecedores de serviços. Há uma relação biunívoca entre o conjunto de fornecedores a si associados e os executores desse bloco, ou seja, por cada fornecedor há um e um só executor.

Visto do lado do mediador todos os executores são iguais. Quer isto dizer que os métodos disponibilizados pelos executores e a forma como devolvem os resultados são os mesmos, independentemente dos fornecedores a que estão associados.

É assim criada uma camada de abstracção em relação aos fornecedores e às particularidades da interacção com eles. Estas particularidades ficam exclusivamente a cargo dos executores que com eles lidam. O executor é assim a única entidade responsável pelo estabelecimento da conexão com o fornecedor e a única que precisa de conhecer a sua `API`, quer em termos da formulação dos pedidos, quer da recepção das respostas.

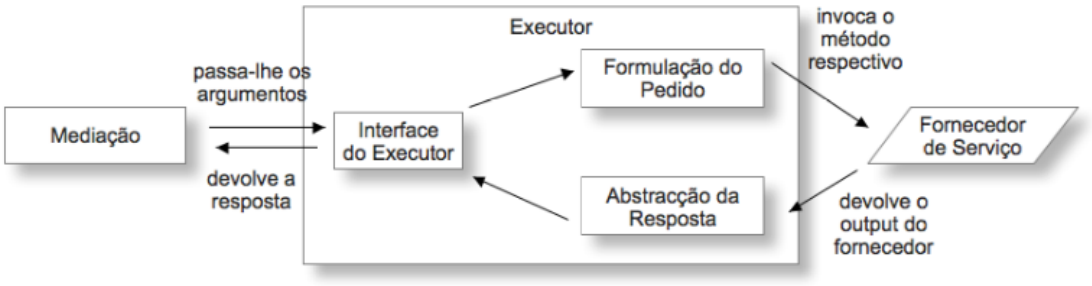

Figura 5.8: Modo de execução do processo de abstracção de resultados

Este facto é ilustrado na Figura 5.8. Um pedido que chegue ao executor é passado ao módulo de formulação de pedido que o transforma num pedido ao fornecedor, no formato deste. A resposta passa por um módulo de abstracção antes de ser devolvido ao mediador.

Na interacção entre mediador e executor nada se passa de acordo com a `API` do fornecedor.

O "enclausuramento" do fornecedor numa caixa preta, o executor, permite também uma separação entre a tecnologia usada para desenvolver o sistema de mediação e as tecnologias usadas pelos fornecedores. Apenas dentro do executor a tecnologia usada pelo fornecedor tem, eventualmente, de ser considerada.

Formulação do pedido

Cada método disponibilizado pelo executor (disponibilizado para cima, ao mediador) obriga obviamente à invocação de pelo menos um dos métodos disponibilizados pelo fornecedor. O executor tem assim de transformar o método e os parâmetros de serviço que o acompanham em pedidos ao fornecedor de acordo com a `API` deste.

Abstracção da resposta

Se um pedido a um fornecedor for bem sucedido, o executor irá receber uma resposta formatada de acordo com a `API` do fornecedor. No entanto, devido a cada fornecedor incluído no sistema possuir a sua própria lógica de negócio, em algumas situações, a resposta a um pedido executado pode ser estruturalmente diferente de fornecedor para fornecedor, independentemente de devolverem o mesmo tipo de conteúdos.

Por isso, torna-se necessária a presença de um processo de abstracção que logo após a execução do pedido ao serviço, seja capaz de criar uma resposta uniforme com base nos resultados obtidos para posteriormente serem devolvidos ao utilizador. Estes blocos de abstracção, como referido na concepção do modelo, não são obrigatórios porque dependem exclusivamente do tipo de serviço em questão. No entanto, se efectivamente necessário, a cada executor existente terá que ser adicionado um módulo de abstracção.

No sentido de construir a abstracção é necessário que o bloco concebido seja capaz de transformar os dados contidos na resolução do pedido numa estrutura intermédia. Esta estrutura intermédia terá que ser concebida de igual forma para todas as abstracções implementadas sendo que o único processo individual será o de "parsear" os resultados para a estrutura.

Assim, torna-se necessário que no processo de concepção da estrutura intermédia, todos os métodos sejam analisados assim como todos os possíveis resultados dos mesmos. Desta forma, esta implementação torna-se algo extensa e complexa podendo ser atenuada apenas devido à existência de traços comuns entre os serviços dada a sua orientação a um tipo de serviço especifico.

5.3 Bloco de Monitorização

O bloco de monitorização tem como principal função recolher dados estatísticos sobre os fornecedores de serviços com o objectivo de capacitar o sistema de mediação de meios nos quais se possa basear aquando da eleição do melhor fornecedores de serviço disponível.

Estes dados serão reunidos com base numa análise focalizada sobretudo em parâmetros relativos às comunicações ou através de características descritivas presentes nas `APIs` dos serviços. Parâmetros relativos às condições ou mesmo características dos serviços, apesar de essenciais para a eleição do serviço, só serão possíveis de depreender por intermédio de uma análise aos fornecedores de serviços por intermédio dos administradores da plataforma. Estes valores serão assim posteriormente adicionados à estrutura intermédia, mas permanecendo alheios à execução do bloco de monitorização.

Será responsabilidade do bloco de monitorização obter dados periodicamente por forma a ser possível calcular valores médios utilizáveis (estabelecidos ao mês por exemplo). Para tal, será a aplicação de monitorização que estabelecerá métricas com base nos valores acima descritos por forma a calcular características relativas à qualidade de serviço tais como a disponibilidade dos serviços ou o tempo médio de resposta dos mesmos.

Por fim, todos estes dados serão colocados numa estrutura intermédia (base de dados) acessível pelos blocos de serviços e de monitorização estabelecendo assim o único elo de ligação entre eles. Na Figura 5.9 encontra-se patente todo este processo.

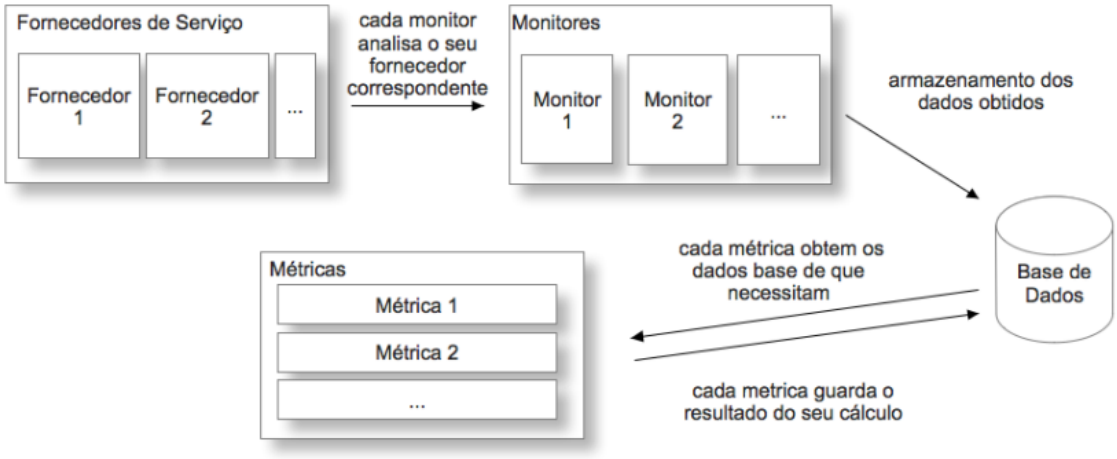

Figura 5.9: Constituição interna do bloco de monitorização

5.3.1 Monitores

Na base do bloco de monitorização encontram-se os monitores. Os monitores tem como principal objectivo efectuar uma análise contínua a determinados parâmetros relevantes

num contexto de utilização dos serviços por forma a ser possível efectuar uma avaliação relativa ao funcionamento dos fornecedores de serviços. Será com base nesta avaliação que o sistema efectuará a selecção do fornecedor a invocar.

Neste sentido, cada monitor dirá respeito exclusivamente a um fornecedor e, como tal, cabe-lhe apenas a si definir os mecanismos necessários face a esse fornecedor para recolher os dados pretendidos.

No entanto, devido à futura necessidade de comparar os valores, é possível especular que determinados mecanismos possam produzir valores com base em modos de execução diferentes e originando margens de erro nos próprios valores. Como tal, é recomendável que os mecanismos implementados na avaliação de um determinado fornecedor sejam equivalentes aos dos restantes fornecedores de serviço do mesmo tipo.

Quando lançado, o monitor irá efectuar operações dirigidas ao fornecedor de serviço correspondente relegando sobre cada operação a análise a um determinado parâmetro ou característica.

Deste modo, cada operação incluída terá a tarefa de obter resultados do serviço que tanto pode ser relativo ao sucesso de uma invocação ou à recolha de valores concretos. Estes parâmetros analisados serão essencialmente relativos às comunicações com os serviços, tais como latência ou tempo de resposta.

Por sua vez, os dados obtidos serão colocados numa base de dados inerente à plataforma por forma a serem acedidos no cálculo das métricas de serviço ou na eleição do melhor serviço disponível.

5.3.2 Métricas

No contexto do bloco de monitorização, os módulos relativos às métricas possuem como principal objectivo executar cálculos que permitam obter valores indicativos da qualidade do serviço em análise.

Para tal, cada métrica possui estabelecida uma fórmula pré-definida normalmente baseada num parâmetro de qualidade de (`QoS`), que irá ser executada com base em valores armazenados na base de dados da plataforma. Estes valores serão por sua vez colocados na base de dados pelos monitores, tornando-se obrigatória a execução destes anteriormente ao cálculo das métricas.

Em algumas métricas, as fórmulas limitam-se a estabelecer uma média dos valores de monitorização, fixando-se apenas num parâmetro de monitorização para o cálculo da métrica. Normalmente, este cálculo é efectuado com base em valores dos últimos 30 dias embora este número dependa exclusivamente da métrica em questão e do administrador

do sistema.

Por norma, as métricas são orientadas a um parâmetro em particular, como por exemplo largura de banda, mas englobam vários serviços disponíveis. Isto acontece devido aos valores base que, independentemente do fornecedor ao qual se relacionam, possuem a mesma forma. Por exemplo, os valores de monitorização que analisam se um fornecedor de serviço se encontra online podem diferir mas o resultado final não variará de positivo ou negativo. Pode acontecer no entanto que, devido à orientação da métrica esta tenha que ser direccionada a um tipo de serviço em particular.

No seu término, os valores obtidos serão colocados, tal como acontece com os valores relativos aos monitores, na base de dados intermédia por forma a poderem ser consultados pela plataforma.

Capítulo 6

Implementação de um Sistema de Mediação

O modelo arquitectural apresentado define-se como um guião através da qual pode ser obtido um sistema de mediação que terá por base os blocos explicados. Neste capítulo será exemplificada a construção de um sistema que se baseie na metodologia apresentada. Saliente-se que o que será descrito pretende ser apenas uma prova de conceito do modelo arquitectural e que como tal não envolverá a totalidade dos aspectos relativos à implementação de um sistema de mediação concreto.

Assim, por forma a demonstrar a capacidade e modularidade da arquitectura na construção de um sistema de mediação, foram escolhidas quatro áreas distintas de serviços, nomeadamente serviços de mapas, meteorologia, fotos e de *instant messaging*. Nos serviços de mapas e meteorologia foi introduzida concorrência através da implementação de vários fornecedores de serviços. A estrutura de apresentação será semelhante à do capítulo anterior por forma a ser possível descrever os diferentes passos envolvidos na construção dos módulos da arquitectura e tecnologias associadas.

Por fim será apresentado o processo de construção de uma aplicação web que utiliza os serviços disponibilizados pela plataforma com base na noção de mashup de modo a demonstrar a usabilidade do modelo proposto.

6.1 Visão Geral do Sistema

Em conformidade com o modelo arquitectural, a implementação do sistema será baseada em três partes fundamentais com a ressalva que, na implementação, estas partes serão de facto aplicações. Assim, cada bloco anteriormente referido dará lugar no sistema a uma aplicação singular mas com dependências para as restantes aplicações.

Deste modo, o sistema de mediação será constituído por uma primeira aplicação com a missão de receber os pedidos efectuados, e como tal, constituir a `API` do sistema. Esta aplicação terá que invocar uma das aplicações de serviços implementados, ou seja, terá que invocar o mediador destinado a mapas, a meteorologia, a fotos ou a *messaging*. É importante salientar de novo que cada mediador possui um tipo de serviço específico e que, como tal, cada um destes tipos constituirá uma aplicação independente das restantes. Finalmente, haverá uma aplicação responsável pela monitorização.

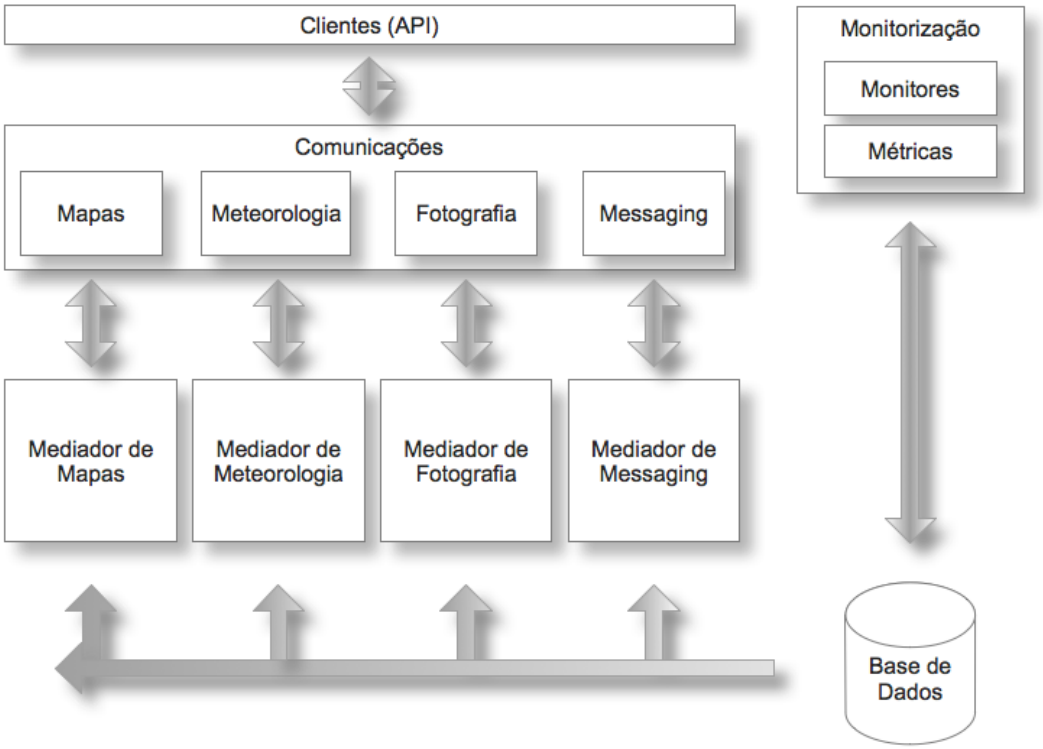

Figura 6.1: Estrutura do Sistema de Mediação Exemplo Implementado

Na Figura 6.1 é ilustrada a disposição dos diferentes elementos no contexto da plataforma assim como as ligações entre os mesmos. Note-se que graças à modularidade da arquitectura, novos tipos de serviços podem ser implementados através da adição de novos módulos orientados ao serviço em causa. De facto, por forma a garantir esta modularidade assim como introduzir desde logo uma elevada capacidade de disponibilização ou uma futura capacidade de expansão, o sistema foi construído de forma a ser utilizado num servidor aplicacional Java, nomeadamente o JBoss [Hat]. No entanto, apenas as aplicações que possam sofrer uma elevada carga foram colocadas no servidor aplicacional o que excluiu desde logo a aplicação de monitorização. Contrariamente às restantes, esta aplicação correrá de forma local tendo como objectivo popular a base de dados do sistema.

6.2 Definição da API

Se o sistema de mediação pretende dar a conhecer as suas funcionalidades, necessita de especificar como podem os clientes aceder aos seus serviços. De facto, como se trata de um sistema invocado remotamente, é necessário que os clientes percebam como podem efectuar um pedido, quais as operações suportadas, quais os seus argumentos e quais os formatos da resposta. Em suma, o sistema necessita de constituir uma `API` para disponibilizar aos potenciais clientes.

O sistema de mediação compreende diversos mediadores, cada um dos quais orientado a um tipo de serviço em particular. De facto, a constituição da `API` da plataforma representa a união das `API`s de cada um dos mediadores. Desta forma, o esforço de concepção da `API` recai sobretudo no tipos de serviço implementados.

A concepção de uma `API` para um determinado tipo de serviço necessita que o implementador comece por analisar todas as operações suportadas pelos fornecedores de serviço incorporados e procure conceber uma lista de todas as operações a implementar. A `API` pode ser dividida em duas partes: uma, vocacionada para uma abordagem global, isto é, orientada aos clientes que pretendam um serviço sem especificar o fornecedor, e uma segunda parte que é redireccionada para um fornecedor em particular.

Como exemplo, atente-se ao tipo de serviço "meteorologia" constituído por dois fornecedores, o *GoogleWeather* e o *YahooWeather*. Este tipo de serviço possui correntemente dois fornecedores implementados. A análise efectuada aos dois serviços depreende apenas uma operação fundamental, a obtenção das condições meteorológicas. No entanto, após um estudo aos dois fornecedores, foi depreendido que um deles é capaz de fornecer informação meteorológica não só para o presente dia, mas também para os três seguintes.

Ignore-se a implementação destes fornecedores no sistema. Se efectuada apenas uma breve análise aos dois fornecedores, é perceptível desde logo a existência de formatos diferentes de devolução de resultados (aspecto abordado mais à frente) assim como a presença de funcionalidades de serviço distintas. De facto, a definição da `API` terá que ser capaz de transmitir estas funcionalidades, ou seja, é necessário que o cliente possa indicar que funcionalidades pretende. Atente-se a algumas funcionalidades relativas apenas a um fornecedor patentes no caso do serviço meteorológico:

- indicação da pressão atmosférica — disponível apenas no *YahooWeather*.
- indicação das condições atmosféricas para os três dias seguintes — disponível apenas no *GoogleWeather*.

- indicação das horas relativas ao nascer/pôr do sol — disponível apenas no *YahooWeather*.

Como é visível, duas destas funcionalidades são exclusivas ao *YahooWeather* enquanto a outra é exclusiva ao *GoogleWeather*. A `API` para o serviço meteorológico terá que, para além de implementar todas as operações disponíveis pelos fornecedores de serviço, incluir como parâmetros destas operações estas funcionalidades. A intenção é que deste modo, o cliente possa indicar na sua invocação ao sistema que os resultados que pretende devem incluir, por exemplo, a indicação da pressão atmosférica.

Além destas funcionalidades, é necessário que a invocação ao serviço inclua os parâmetros necessários para a execução do próprio método, neste caso, a indicação da localização de onde é pretendida a informação meteorológica. Por forma a usufruir das capacidades de monitorização do sistema de mediação, o cliente pode também incluir na sua invocação, a indicação de uma qualidade de serviço que pretenda, como por exemplo, uma elevada disponibilidade.

Assim, a declaração de um método de serviço no sistema de mediação, compreende na `API` três tipos de parâmetros: *parâmetros do método, parâmetros relativos às funcionalidades do serviço* e *parâmetros de monitorização*. Ao associar estes factores a um método específico presente no sistema, "obterCondiçõesMeteorológicas", a declaração (tendo em conta apenas os aspectos referidos) assume na `API` a seguinte forma (simplificada):

- *Serviço* — Meteorologia
- *Operação* — "obterCondiçõesMeteorológicas"
- *Parâmetros Método* — localização (*String*)
- *Parâmetros Funcionalidades de Serviço* — pressão atmosférica (*boolean*); horas sol (*boolean*); condições dos próximos dias (*boolean*)
- *Parâmetros Monitorização* — disponibilidade (*boolean*)
- *URL* — "http://medsys/weather/obterCondicoesMeteorologicas"
- *Formato da Resposta* — Texto (*String*)

Neste caso, com base nesta declaração, o cliente efectua um pedido ao sistema no sentido de obter as condições meteorológicas para uma localização, e indica-lhe os parâmetros que pretende ver cumpridos. No entanto, denote-se que nenhuma referência é efectuada ao fornecedor pretendido. De facto, como anteriormente referido, não faz sentido indicar ao sistema características relativas à eleição do fornecedor se o cliente indica qual pretende. No entanto, se o cliente pretende de facto um fornecedor específico, então o método utilizado não será o mesmo. A declaração da `API` para o mesmo método direccionado a um fornecedor especifico (*GoogleWeather*) fica da seguinte forma:

- *Serviço* — Meteorologia
- *Operação* — "obterCondiçõesMeteorológicas"
- *Parâmetros Método* — localização (*String*)
- *URL* — "http://medsys/weather/GoogleWeather/obterCondicoesMeteorologicas"
- *Formato da Resposta* — Texto (*String*)

Desta forma, cada operação presente num tipo de serviço compreende duas declarações, uma direccionada a um fornecedor genérico e outra redireccionada a um fornecedor específico. A `API` global por sua vez, é composta por todas as operações presentes nas diversas `APIs` dos tipos de serviços implementados no sistema.

De modo a facilitar a utilização da plataforma, em adição à `API` genérica implementada, foi constituída uma `API` em Java sob a forma de uma biblioteca, que encapsula a utilização da plataforma como um serviço web, permitindo aos programadores a rápida incorporação dos serviços da plataforma nas suas aplicações. Esta biblioteca, assente sobre a `API` genérica, pretende que futuros programadores não necessitem de estudar como implementar web services para utilizar os serviços do sistema de mediação, tornando assim mais abrangentes as possíveis utilizações da plataforma. Assim, para utilizarem o sistema de mediação, os programadores apenas necessitam de importar a biblioteca correspondente ao tipo de serviço pretendido e, com base na documentação fornecida na mesma, invocar os métodos correspondentes.

6.3 Bloco de Comunicações

O bloco de comunicações corresponde à entidade com quem os clientes interagem. Como tal, é através deste bloco que a requisição de conteúdos ao sistema é efectuada. Assim, quando um pedido é enviado à plataforma, este será recepcionado pelo bloco de comunicações e posteriormente redireccionado para um dos mediadores de acordo com o tipo de serviço pretendido. Deste modo, o bloco de comunicações estabelece-se como o responsável pela implementação da `API` do sistema.

Como explicado no Capítulo 5, o bloco de comunicações é composto por dois tipos de módulos principais, o *router* (singular) e os *resources*, com o objectivo de constituir o serviço web do sistema. No entanto, antes de ser efectuada a sua elaboração, é necessário eleger uma tecnologia de implementação de serviços, ou seja, eleger entre `REST` e `SOAP`. Neste caso, `REST` foi eleito devido à sua simplicidade.

A implementação de um serviço Web em `REST` depreende várias necessidades, a primeira das quais relacionada com a associação de operações a URLs específicos. De facto

é através de URLs, que os clientes podem efectuar pedidos à plataforma. No entanto, é necessário que o sistema possua mecanismos que os interprete por forma a perceber que operação é requisitada. Por exemplo, um pedido a um serviço de mapas será diferente de um pedido a um serviço meteorológico, mas é necessária uma capacidade do sistema para depreender isso mesmo. O módulo Router pretende resolver este problema através da análise aos URLs recebidos.

Router

Na secção anterior foi explicado que a constituição da API do sistema é elaborada essencialmente através da união das APIs de todos os mediadores existentes. O módulo Router distingue esta associação. De facto, como explicado no Capítulo 5, cada Resource é concebido face a um tipo de serviço em particular. O Router, através da análise do URL recebido, reencaminha o pedido para o Resource correspondente. No entanto para que tal suceda, é necessário que a composição dos URLs a utilizar depreenda este mesmo facto.

Segundo o conceito arquitectural de REST, os URLs devem ser auto-descritivos. Deste modo, foi definido que por forma a ser perceptível para o sistema qual o tipo de conteúdo que é requisitado, a primeira variável na constituição do URL indicaria qual o tipo de serviço invocado. Assim, no contexto dos tipos de serviço requisitados, existem quatro entradas na plataforma:

- *Mapas* - Através do endereço `http://medsys/maps/...`
- *Messaging* - Através do endereço `http://medsys/messaging/...`
- *Fotos* - Através do endereço `http://medsys/photos/...`
- *Meteorologia* - Através do endereço `http://medsys/weather/...`

Todos os pedidos que alcancem a plataforma e não possuam um destes formatos são desde logo descartados. Os que possuem, são reencaminhados para resources correspondentes. No entanto, o *parsing* do Router não compreende apenas os tipos de serviço. Como referido anteriormente, podem existir casos em que o cliente pretende um fornecedor de serviço específico. Nestes casos, a solução encontrada é simples. Após a indicação do tipo de serviço que pretende, o cliente pode de seguida indicar um fornecedor específico, constituindo o URL, por exemplo, da seguinte forma: `http://medsys/maps/GoogleMaps/...`. O procedimento é em tudo igual ao caso genérico com a diferença que o pedido será enviado para um módulo Resource próprio dada a especificação apresentada.

A constituição dos URLs termina com a definição das operações. Um serviço pode conter mais que uma operação e como tal é necessário que o sistema perceba qual a operação requisitada. Deste modo, a parte final da constituição de um `URL` compreende este aspecto. Por exemplo, a constituição básica de um pedido por um novo mapa do GoogleMaps é efectuada através do URL `http://medsys/maps/GoogleMaps/createMap/`. Note-se que o termo relativo à especificação do fornecedor "GoogleMaps" é facultativo enquanto que a operação elegida não o é. A constituição de um URL representativo deste aspecto seria `http://medsys/maps/createMap/`.

Resources

Os Resources representam a ligação entre o bloco de comunicações e os blocos de mediação. Simultaneamente, definem-se também como a primeira fase de análise aos pedidos. De facto, tal como acontece com a arquitectura `REST`, cada Resource pretende assumir-se como uma representação de um determinado conteúdo. Por exemplo, Resource "*PhotoResourceAlbum*" refere-se a álbuns existentes nos serviços de fotos e aos quais o cliente pode aceder. No entanto, se o cliente pretender obter fotografias então terá que utilizar um Resource representativo destas (neste caso, "*PhotoResourcePhotos*") devido a tratar-se de um conteúdo diferente.

Por forma a simular este conceito, cada Resource pode conter no máximo quatro métodos, constituídos com base nas operações HTTP *get*, *post*, *put* e *delete*. De facto, cada método implementado tem que ter por base uma destas operações. Por norma, em `REST`, estas operações representam operações CRUD[1], o que facilita a eleição da operação a utilizar de acordo com o objectivo do método. No entanto, um Resource não necessita de implementar a totalidade das operações, sendo usual a utilização apenas de operações *get* e *post*.

Atente-se à implementação do serviço de messaging na plataforma, nomeadamente o fornecedor *Twitter*. A constituição do serviço de messaging é composta, entre outros, pelos seguintes métodos definidos na `API` do sistema (ver Apêndice A):

- *update_status* - Envia uma nova mensagem de perfil.
- *getFriendsTweets* - Obtém uma lista com todas as mensagens dos seus amigos.
- *sendPrivateMessage* - Envia uma mensagem exclusivamente a um amigo.
- *receivePrivateMessage* - Obtém todos as mensagens enviadas ao utilizador de forma privada, ou seja, exclusiva.

[1] Create, Read, Update and Delete — funções básicas de armazenamento.

Neste contexto, é possível verificar que algumas destas operações assumem um perfil para obtenção de um conteúdo (get), enquanto outras assumem um perfil de criação de um conteúdo (post). Concretamente, as operações *update_status* e *sendPrivateMessage* representam pedidos post, enquanto as restantes representam pedidos get. No entanto, os conteúdos a que se referem são diferentes. Por exemplo, as funções *sendPrivateMessage* e *receivePrivateMessage* referem-se a um único conteúdo, neste caso, a mensagens privadas. As restantes diferem. Neste sentido, a implementação destas operações compreenderá três Resources.

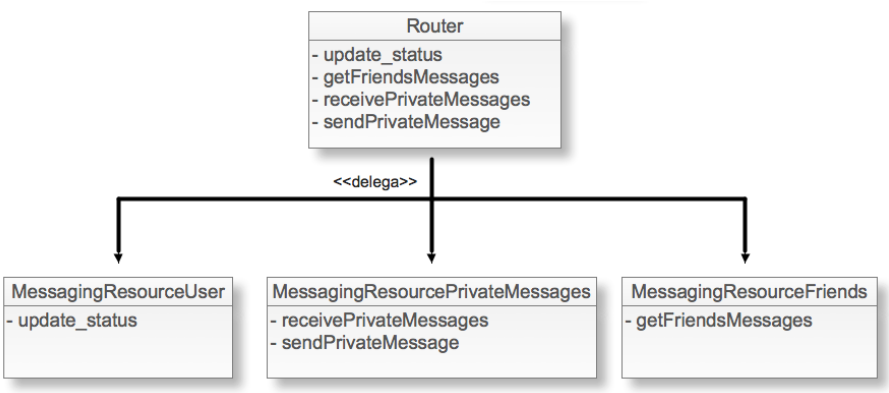

Figura 6.2: Exemplo de Implementação do bloco de comunicações

A Figura 6.2 ilustra a estrutura do bloco de comunicações com base no exemplo apresentado. O método *update_status* pretende actualizar o perfil do utilizador. Os métodos *receive* e *sendPrivateMessage* pretendem efectuar operações sobre mensagens privadas. Por fim, o método *getFriendsMessages* pretende obter mensagens relacionadas com os amigos do utilizador. Com base nestas interpretações, torna-se necessária a criação de três Resources, como apresentado na figura, nos quais o módulo Router possa delegar os pedidos recebidos.

No caso apresentado, para o serviço de *messaging*, apenas se encontra disponível um fornecedor. No caso de existirem mais fornecedores, o conjunto de Resources terá também que reflectir este aspecto. Por cada Resource constituído no sistema, terá que ser feita uma replicação do mesmo. Por exemplo, atente-se ao bloco de comunicação na parte referente à criação de mapas, apresentado na Figura 6.3. Ele apresenta a diferenciação patente na camada de Resources no momento de recepção de um pedido a um método que possui diversos fornecedores para a sua resolução. Neste caso, o módulo *MapsResource* resolve todos os pedidos que sejam invocados sem a introdução de um fornecedor específico. Se, no entanto, tal acontecer, então o pedido será redireccionado para o módulo *MapResource* tendo em anexo a indicação do fornecedor requisitado.

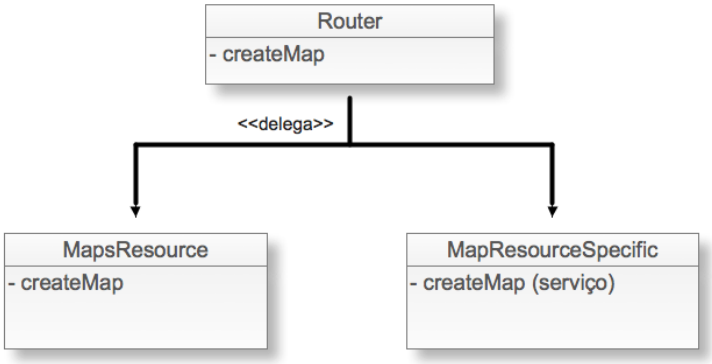

Figura 6.3: Distinção entre pedidos genéricos e pedidos específicos a um fornecedor na constituição do bloco de comunicações

No Capítulo 5 é explicado que, no contexto de um pedido, o cliente pode (se disponível na `API` do sistema), indicar aspectos relacionados com a futura escolha do fornecedor. Estes aspectos podem compreender indicações relacionadas com a monitorização dos serviços ou aspectos relacionados com as suas funcionalidades. No entanto, se o cliente elege à partida qual o fornecedor que pretende, estes aspectos não possuem qualquer relevância. Neste sentido, uma análise à presença destes aspectos apenas é efectuada se, utilizando o exemplo anterior, o cliente simplesmente requisitar um mapa sem especificar um fornecedor em concreto. No contexto de um pedido por um mapa genérico, o cliente pode requisitar alguns dos seguintes parâmetros:

- *Latitude/Longitude* - indicação das coordenadas geográficas onde o mapa se centrará.
- *Tipo de Mapa* - Indicação do tipo de mapa pretendido. Por exemplo, vista satélite, normal ou híbrida.
- *StreetView* - Indicação que pretende um mapa com a funcionalidade "StreetView".
- *Disponibilidade* - Indicação que pretende um mapa proveniente do fornecedor que melhor disponibilidade apresente.

Segundo a `API` do sistema, desta lista de parâmetros, o cliente apenas tem a obrigatoriedade de indicar a latitude e a longitude. Os restantes são facultativos. Denote-se que, no caso de o cliente especificar um fornecedor em particular, os últimos dois parâmetros presentes na lista não possuem qualquer relevância e como tal são descartados pelo *resource* correspondente.

No caso de o cliente não especificar o fornecedor, então o Resource irá mapear todos os parâmetros numa lista que enviará ao mediador de mapas para a sua resolução. Este envio,

no entanto, encontra-se dependente da análise aos parâmetros obrigatórios presente no próprio Resource. No caso particular da criação de um mapa, é obrigatório que o cliente forneça dois aspectos, ou a latitude/longitude, ou o endereço de uma localização. Se um destes requisitos for cumprido, o Resource invocará então o mediador correspondente entregando-lhe a lista de argumentos encontrados. Se um fornecedor for especificado, então será enviada ao mediador a indicação do fornecedor assim como todos os argumentos pertinentes para a resolução do pedido (neste caso, latitude/longitude e tipo de mapa).

A implementação do bloco de comunicações foi efectuada com base na *framework Restlet* [Tec]. A escolha desta framework baseou-se essencialmente na sua capacidade em construir o serviço web da forma mais aproximada possível aos conceitos defendidos pela arquitectura RESTful. Em adição, esta framework permite a disposição dos módulos Resources num estrutura arbórea em função do módulo inicial, o Router, tornando assim possível a análise ao URL recebido e consequente delegação no módulo respectivo. Desta forma, a adição de novos tipos de serviços ao sistema é simplificada, sendo apenas necessária a constituição do módulo Resource a implementar, e sua consequente associação a um `URL` especifico no Router.

6.4 Monitorização

Por forma a dotar o sistema com informação relativa ao estado dos fornecedores de serviço implementados, foi constituído no sistema de mediação, para além dos blocos relativos à mediação dos serviços, uma aplicação de monitorização.

Esta monitorização, efectuada de forma individual a cada fornecedor coberto, permite ao sistema captar dados relativos ao seu desempenho, como por exemplo obter a percentagem de *uptime* do serviço, perceber qual o tempo de resposta a um pedido ou qual a latência apresentada, através de métricas pré-estabelecidas.

Para tal ser possível, é necessário no entanto que, por forma a obter cada um destes aspectos, o processo de monitorização possua um método de execução bem definido. Como tal, foi definido na sua concepção que o processo de monitorização assentaria nas seguintes fases:

- Obtenção dos dados relativos às comunicações.
- Armazenamento persistente dos valores obtidos numa base de dados.
- Estabelecimento de médias relativas aos dados obtidos.

No Capítulo 5 é referido que o bloco de monitorização é constituído essencialmente por dois tipos de módulos, os Monitores e as Métricas. Com efeito, a lógica de negócio da aplicação de monitorização tem por base a recolha (através dos *monitores*) e posterior cálculo de valores (com base em *métricas*) associados aos fornecedores de serviços. No entanto, por forma a conceber uma aplicação concreta é necessário definir alguns aspectos relativos ao seu funcionamento. Um destes aspectos consiste na duração do processo de recolha de dados provenientes de um fornecedor.

No modelo arquitectural, cada Monitor é responsável pela recolha de valores associados a um só fornecedor através de diversas iterações ao serviço. No entanto, o sistema de mediação pode conter um número elevado de fornecedores o que condiciona a duração total do processo de recolha dos dados. De modo a reduzir o tempo total de execução dos monitores, cada Monitor presente na aplicação será executado de forma concorrente relativamente aos restantes.

De facto, a introdução de concorrência não deriva apenas de uma elevada duração do processo de monitorização, o problema resulta da necessidade de repetir este processo ao fim de um determinado intervalo de tempo, tratando-se de um processo cíclico. Deste modo, se a execução dos diversos monitores ocupar, por exemplo um minuto, e se existirem quinze monitores, então o tempo total de execução será de quinze minutos. Este caso poderia dar origem a uma incompatibilidade se o administrador do sistema entender que o processo de monitorização deveria ser efectuado a cada dez minutos.

A Figura 6.4 ilustra a estrutura global da aplicação de monitorização. Devido à existência de concorrência, é necessário que este modelo contenha, para além dos módulos explicados na arquitectura (Monitor e Métrica), um módulo capaz de lançar todos os monitores existentes, o *Invoker*. Este módulo recebe do administrador do sistema a indicação de todos os fornecedores a monitorizar assim como a indicação de quais as métricas que se pretendem calcular. Desta forma, a missão do módulo centra-se em invocar de forma concorrente (através do lançamento de *threads*) um objecto Controller por cada fornecedor indicado pelo administrador do sistema. Ao lançar cada *thread*, o Invoker associa-lhe ainda a informação relativa ao fornecedor a monitorizar assim como quais as métricas a calcular.

Suponha-se um caso em que o administrador do sistema pretende obter dados de monitorização relativos a dois fornecedores, o *"GoogleMaps"* e o *"YahooMaps"*. Em adição, o administrador pretende ainda que seja calculada a disponibilidade relativa aos dois fornecedores. Neste caso, a execução do Invoker necessitará de criar duas *threads*, uma associada ao *"GoogleMaps"* e a outra ao *"YahooMaps"*, indicando-lhes que pretende obter o cálculo da disponibilidade relativa ao fornecedor associado.

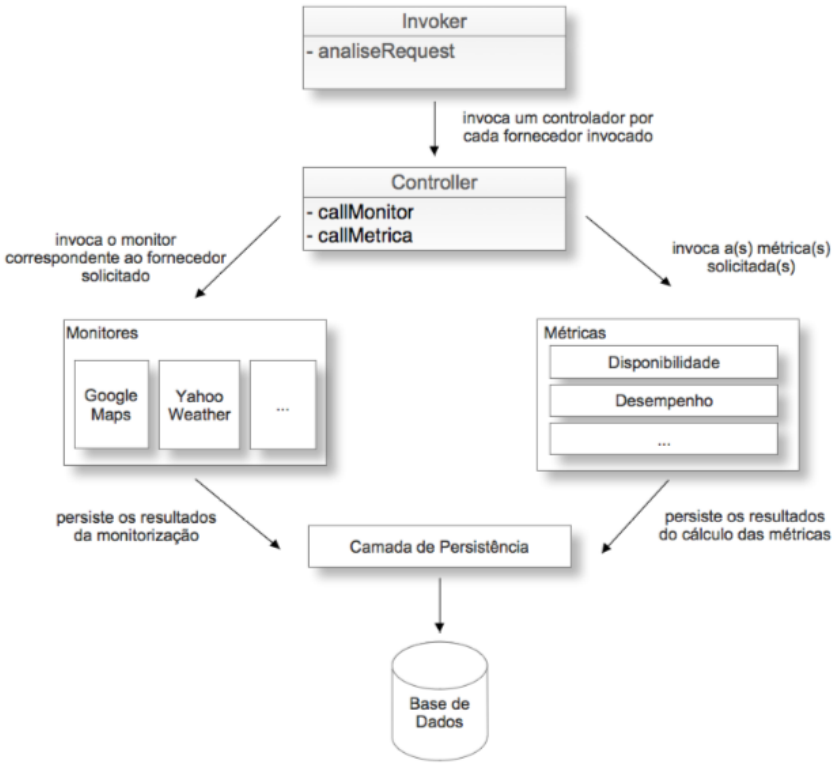

Figura 6.4: Estrutura global da aplicação de monitorização

A Figura 6.5 mostra o processo de execução da aplicação de monitorização com base no pedido efectuado pelo administrador. Após a invocação das duas *threads*, cada uma destas correrá separadamente, de acordo com os fornecedores de serviço a si associados. Cada *thread* invocará o monitor relativo ao seu fornecedor armazenando os resultados na base de dados do sistema. De forma a ser efectuado o cálculo da disponibilidade é necessário antes de mais perceber como será efectuada a obtenção dos dados através dos monitores do sistema.

Obtenção de dados relativos às comunicações

Como referido anteriormente, a tarefa de obter dados relativos às comunicações é, no sistema de mediação, da responsabilidade dos Monitores. Cada Monitor é construído em função de um fornecedor em particular implementando mecanismos que tornem possível (em relação a esse fornecedor) a recolha dos dados pretendidos. Como tal, concebeu-se a estrutura de um *monitor* como algo que implementa mecanismos por forma a obter dados relativos a um fornecedor independentemente de ser possível ou não, que com estes mesmos mecanismos se possa obter dados relativos a outros fornecedores. Neste sentido, cada *monitor* opera segundo uma lógica própria. O propósito de um *monitor* centra-se

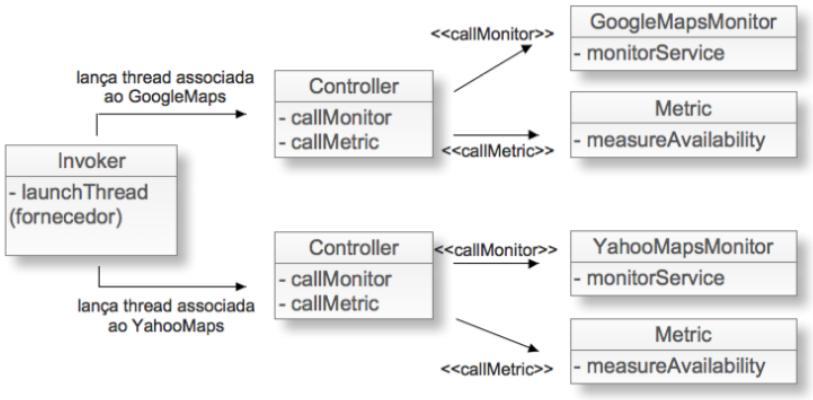

Figura 6.5: Exemplo do processo de execução na aplicação de monitorização

em recolher dados relativos às comunicações de um dado fornecedor sendo a escolha das ferramentas ou métodos a utilizar completamente irrelevante para o sistema. A decisão de como recolher os dados dos fornecedores depende apenas do administrador, com a condição de que, no término de cada iteração, os dados sejam colocados na base de dados do sistema.

No contexto do sistema de mediação, foram implementados cinco monitores relativos aos fornecedores *GoogleMaps*, *YahooMaps*, *BingMaps*, *GoogleWeather* e *YahooWeather*. No sentido de demonstrar o processo de recolha de dados, será utilizado o fornecedor *"YahooMaps"*.

Por forma a exemplificar o processo de monitorização do *YahooMaps* é necessário que, antes de serem estabelecidas as ferramentas a utilizar, seja definido um conjunto de parâmetros a monitorizar no contexto das comunicações entre os fornecedores e o sistema de mediação. Assim, foram definidos cinco parâmetros: *latência*, *tempo de resposta*, *estado do fornecedor*, *largura de banda* e *round-trip-time*.

A obtenção destes parâmetros é efectuada segundo operações diferentes. No caso da verificação do *estado do fornecedor*, o processo resume-se a verificar se o fornecedor se encontra activo. Para tal, o monitor apenas se liga ao *YahooMaps* através do protocolo *HTTP* e procura obter o código de resposta do mesmo. Se o código obtido for 200 então o serviço encontra-se activo, e neste caso o monitor criará na base de dados uma entrada relativa a este valor.

A obtenção da latência e do tempo de resposta do fornecedor já necessitam de envolver a utilização de uma ferramenta em particular, o *echoPing* [Bor]. O *echoPing* é uma ferramenta simples, executada através da linha de comandos, que perite obter dados relativos a servidores remotos através do envio de pedidos, tais como pedidos HTTP.

```
This is echoping, version 5.2.0.

Trying to connect to internet address 212.113.163.11 80 to transmit 88 bytes...
Trying to send 256 bytes to internet address 212.113.163.11...
Connected...
TCP Latency: 0.023469 seconds
Sent (88 bytes)...
Application Latency: 0.577173 seconds
```

Figura 6.6: Exemplo de execução do comando *echoping*

A Figura 6.6 apresenta um exemplo de utilização da ferramenta via linha de comandos demonstrando um pedido executado a um servidor com o objectivo de obter a latência do serviço. A determinação do tempo de resposta é obtida com base num processo análogo. É perceptível na figura que a ferramenta obtêm os valores de latência com base no envio de um número definido de bytes. Saliente-se apenas que, aquando da obtenção de valores relativos a latência noutros fornecedores, este número não se alterou de forma a possibilitar a futura comparação entre fornecedores. Devido à execução da ferramenta ocorrer na linha de comandos, por forma a ser incorporada na aplicação de monitorização, foi necessário que após o término da sua execução, fosse executado um parsing aos resultados obtidos para posteriormente poderem ser armazenadas.

Os restantes parâmetros são obtidos com base em processos semelhantes. O cálculo de dados relativos ao *round-trip-time* foi executado com base no comando *ping*. Apesar de este comando se mostrar por vezes ineficaz na obtenção de resultados face a alguns fornecedores de serviço, no caso do *YahooMaps* isto não se verificou, e como tal, a sua utilização foi incorporada na aplicação através de um processo de parsing semelhante ao utilizado com a ferramenta anterior (*echoping*).

O último dos parâmetros definidos é o cálculo da largura de banda. Este cálculo procura perceber onde se situa o estrangulamento nas comunicações entre o serviço (*YahooMaps*) e o sistema de mediação. No entanto, neste caso, este cálculo revelou-se inútil devido à limitação da ligação utilizada para contactar o serviço, forçando a que o *bottleneck* se situasse no computador de testes. Como tal, este parâmetro foi ignorado na implementação da aplicação.

Finalizado o processo de obtenção dos valores é apenas necessário que os monitores guardem os dados encontrados numa base de dados por forma a serem posteriormente utilizados no cálculo das métricas.

Armazenamento persistente dos valores

No contexto do sistema de mediação, tornou-se necessária a existência de uma base de dados para que a escolha de um fornecedor possa ser efectuada (dependendo do pedido do

cliente). Neste sentido, é necessário que a base de dados constituída contenha informação relativa não só à obtenção dos dados relacionados com as comunicações mas também dados relativos ao cálculo das métricas.

Para tal, cada entidade presente na base de dados corresponde a um determinado parâmetro ou métrica de serviço monitorizável, na qual a aplicação de monitorização pode guardar os resultados da sua operação. Por forma a identificar o serviço monitorizado, cada entrada na base de dados terá ainda associado o fornecedor correspondente assim como a data relativa à operação.

Posteriormente, será com base em alguns destes dados que o cálculo das métricas será efectuado. Sobre a base de dados construiu-se uma camada de persistência à qual os restantes módulos se possam conectar por forma a garantir maior segurança no armazenamento dos dados.

Cálculo de métricas

Uma métrica no contexto do sistema de mediação é um cálculo efectuado com base em dados de monitorização que permite perceber a capacidade de um determinado fornecedor relativamente a um determinado aspecto. Estes aspectos referem-se fundamentalmente a parâmetros `QoS` relativos aos fornecedores de serviço. Na implementação do sistema, incluíram-se três destes aspectos: disponibilidade, desempenho e tempo médios (de monitorização), já explicados no decorrer do Capítulo 4.

Cada métrica possui uma fórmula de cálculo específica e como tal necessita de um processamento distinto relativamente a cada uma das restantes. Neste sentido, por cada métrica incorporada no sistema, existe na aplicação de monitorização um módulo próprio. No entanto, apesar da especificidade do cálculo, é possível dividir o processo de execução em três fases:

- obtenção dos dados relativos à monitorização.
- cálculo da métrica através da fórmula correspondente.
- guardar o resultado na base de dados do sistema.

Por forma a descrever este processo, será exemplificado o cálculo da disponibilidade média de um fornecedor em relação ao último mês. Este cálculo resulta da percentagem total que o serviço esteve operacional.

Por forma a calcular a disponibilidade é necessário em primeiro lugar um levantamento dos dados colocados na base de dados relativos ao estado do fornecedor no último mês.

De seguida é contabilizado o número total de entradas obtidas pelo monitor assim como o número total de valores negativos, ou seja, falhas de serviço. Após este passo, o calculo da métrica pode ser efectuado com base na fórmula correspondente. Neste caso, a fórmula de cálculo é:

$$Disponibilidade = 1 - \frac{n^o\ total\ de\ falhas}{n^o\ total\ de\ entradas}$$

Após a realização do cálculo, resta apenas guardar o valor encontrado na base de dados do sistema através da utilização da camada de persistência.

As restantes métricas possuem um processo de execução semelhante diferindo apenas na fórmula de cálculo. Por exemplo, o cálculo relativo ao desempenho é efectuado com base na média dos tempos de resposta obtidos pelos monitores. Na sua execução são retirados os dois melhores e os dois piores valores por forma a resultar numa avaliação de desempenho mais objectiva.

Interface da Aplicação

Por forma a fornecer ao administrador um método mais acessível de lançamento da monitorização foi concebida uma interface gráfica para a aplicação. Construída através de bibliotecas Java, esta interface permite ao administrador seleccionar quais os fornecedores que pretende monitorizar assim como indicar os parâmetros e as métricas às quais será efectuada a monitorização. Adicionalmente, a interface permite ainda a indicação do número de execuções pretendido e qual o intervalo de tempo entre execuções (ver Figura 6.7).

O processo de execução da aplicação consiste na invocação contínua do módulo *invoker* de acordo com o número de ciclos e intervalo de tempo especificados. No seu término, a aplicação indica ao utilizador os resultados de monitorização obtidos.

6.5 Bloco de Serviços de Mediação

No início do Capítulo 5 designou-se o sistema de mediação como uma entidade multi-mediadora capaz de agrupar sobre si diversos fornecedores reunidos pelo seu tipo de serviço. De facto, são os blocos de serviço de mediação que lhe conferem esta mesma designação. Cada bloco de serviços de mediação é orientado a um tipo de serviço em particular. Na implementação do sistema de mediação, como referido anteriormente, foram criados quatro blocos de serviços de mediação, nomeadamente referentes ao tipo de serviço de mapas, de meteorologia, de fotografia e de *messaging*. Esta secção focalizar-se-á

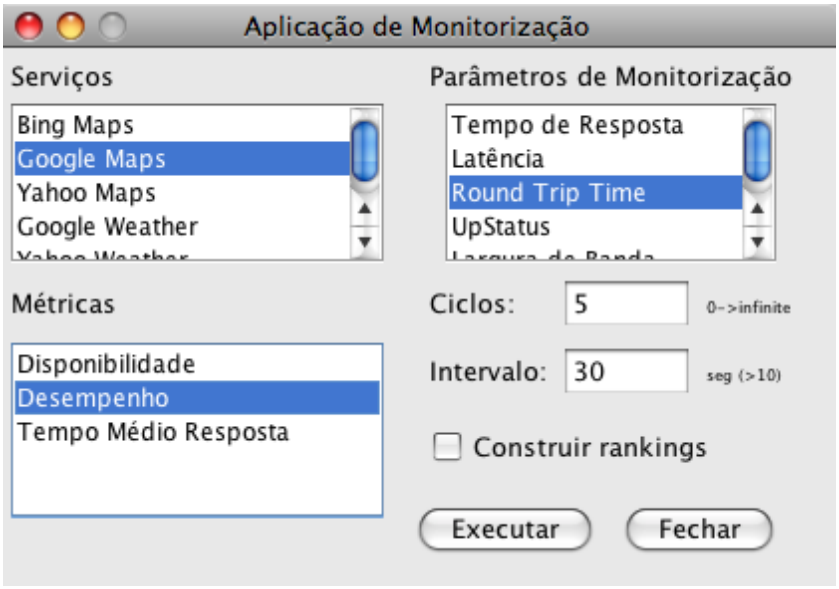

Figura 6.7: Interface Gráfica da Aplicação de Monitorização

na implementação de dois, os serviços de mapas e de meteorologia, fazendo-se uma breve referência aos restantes no final da secção.

6.5.1 Mediador para Meteorologia

O serviço de meteorologia pretende fornecer aos seus clientes informação sobre as condições meteorológicas numa dada localização. No entanto, de forma a implementar o serviço é necessário um levantamento relativo aos elementos presentes neste tipo de serviço. Este levantamento pretende perceber o processo de resolução de um pedido no contexto do serviço meteorológico. Após o término desta análise pode então o serviço ser adaptado ao modelo criado por forma a incorporar estes conteúdos no sistema.

Análise do Serviço

A análise no contexto do serviço de meteorologia será efectuada com base nos fornecedores de serviço a implementar, o *GoogleWeather* e o *YahooWeather*.

O *GoogleWeather* opera de uma forma relativamente simples. A sua invocação necessita que a localização seja colocada segundo o formato (Local,País). O formato de resposta compreende um documento `XML` segundo uma estrutura própria. Nesse documento são colocados aspectos como a condição do tempo, a temperatura, a humidade ou a velocidade do vento para o presente dia, e as temperaturas mínimas e máximas para os dias seguintes. Se for impossível para o serviço determinar uma localização, então devolve uma mensagem de erro segundo o mesmo formato.

O *YahooWeather* possui um processamento algo distinto. O seu formato de invocação requer que o cliente introduza um código de área. No entanto, este código não representa um formato universal (como código postal) mas uma representação estabelecida pela própria Yahoo. O formato de resposta é, à semelhança do Google, um documento `XML`, envolvendo os mesmos aspectos que o anterior com a adição da indicação da pressão atmosférica e horas relativas ao pôr/levantar do sol. No entanto, apenas é capaz de prever as condições meteorológicas para o presente dia e dia seguinte.

Havendo formatos de invocação distintos entre os dois fornecedores foi necessário estabelecer-se o formato ou formatos a usar pelo sistema de mediação. Optou-se por usar o formato (Local,País) de forma a simplificar a invocação dos serviços por parte dos clientes dado tratar-se de um formato mais natural.

É com base nesta análise que são constituídas as funcionalidades de serviço presentes no mediador. Neste caso, repare-se que existem semelhanças entre os dois fornecedores mas, no entanto, existem também algumas diferenças. De facto, podem acontecer situações em que o cliente pretende um serviço que contenha uma destas características exclusivas. Neste sentido, é constituída na base de dados do sistema, uma tabela onde é descrita uma funcionalidade de serviço com a respectiva associação ao fornecedor que a permite. Por exemplo, neste caso, a indicação da pressão atmosférica encontra-se apenas associada ao *YahooWeather*. Se um cliente necessitar desta característica em particular, pode indicá-la no pedido, e posteriormente, durante a avaliação do melhor fornecedor, a característica irá ser considerada. Em adição, para além de incluir a funcionalidade na base de dados do sistema, esta irá ser constituída como um possível parâmetro de invocação na `API` do sistema.

Elaboração do bloco de mediação de serviços

A construção de um bloco de mediação para serviços requer, segundo o modelo arquitectural explicado no Capítulo 5, a construção dos seguintes módulos: *Controlador*, *Analisador*, *Best Provider*, *Mediador Intermédio* e *Executores*.

A implementação do serviço meteorológico compreendeu quatro operações mas por forma a exemplificar o processo de construção do mediador, iremos focalizar-nos apenas em uma. A operação escolhida pretende obter as condições meteorológicas, no formato `XML` e em graus Celsius, para uma dada localização ("*obterCondiçõesMeteoXML*"). Assuma-se ainda que a operação já se encontra definida no bloco de comunicações e que existe um *resource* que irá invocar este mediador. O primeiro elemento a constituir é o *controlador*. Nele é definida a operação ("*obterCondiçõesMeteoXML*") como um dos métodos do medi-

ador que poderá ser invocada pelo bloco de comunicações. Como argumento, este método recebe uma lista de todos os parâmetros invocados pelo cliente assim como a indicação, se existente, do fornecedor requisitado.

De seguida, é necessário conceber o *analisador* e o *best provider*. O analisador tem como objectivo compor, a partir da lista de parâmetros recebida pelo controlador, dois objectos, um com os parâmetros relativos à invocação do serviço, o outro com os parâmetros relativos à eleição do fornecedor. Por exemplo, propriedades como a localização ou o formato da temperatura são anexadas ao objecto de invocação de serviço; propriedades como desempenho ou indicação da pressão atmosférica são anexadas ao objecto direccionado para a eleição do fornecedor. Neste sentido, o analisador possui dois métodos, cada um destinado a compor um dos objectos. O *best provider* é constituído com base na verificação das funcionalidades de serviço indicadas e aspectos de monitorização genéricos. O seu processo de execução é implementado consoante as indicações presentes no Capítulo 5.

Os restantes módulos à excepção do módulo de abstracção são construídos com base na definição geral. O mediador intermédio implementa as operações disponíveis pelo serviço por forma a invocar cada executor com base na indicação do fornecedor elegido. Por sua vez, cada executor é elaborado com base num fornecedor específico com o objectivo de invocar os métodos da sua `API`. No caso do serviço meteorológico, existem dois executores, um destinado à invocação do *GoogleWeather* e o outro destinado ao *YahooWeather*.

Devido ao modo de invocação do *YahooWeather* ser efectuado com base num código, ao inverso do que acontece com o *GoogleWeather* (localização), isto constitui um problema no sentido de conceber uma forma genérica de invocação para o cliente. Deste modo, por forma a ser utilizada a localização como parâmetro genérico de invocação para serviços de meteorologia, implementou-se no módulo de execução relativo ao *YahooWeather*, uma tradução da localização para o código necessário. Esta tradução foi efectuada através de um serviço externo.

Devido ao modo de invocação do *YahooWeather* ser efectuado com base num código e não no par (Local, País), formato com que chega o pedido, manifestou-se um problema adicional. Deste modo, por forma a ser utilizado o formato indicado pela `API` como parâmetro genérico de invocação para serviços de meteorologia, implementou-se no módulo de execução relativo ao *YahooWeather*, uma tradução da localização para o código necessário. Esta tradução foi efectuada através de um serviço externo.

Por forma a proporcionar ao cliente a possibilidade de requerer um serviço ao sistema sem qualquer preocupação relativamente ao fornecedor que o serviu, é necessário que o

sistema lhe forneça os conteúdos segundo uma representação independente do fornecedor.

No caso do serviço de meteorologia, ambos os fornecedores respondem a um pedido através de um documento `XML` estruturado. Para o cliente ter a noção de transparência, é necessário que o sistema estabeleça um formato único capaz de englobar todos os elementos que poderiam estar presentes nos formatos de resposta dos dois fornecedores.

```
<weather unit="C">
    <info>
        <city>Porto, Portugal</city>
        <date>Tue Sep 29 12:48:41 WEST 2009</date>
    </info>
    <current_weather>
        <temperature>28</temperature>
        <condition>Clear</condition>
        <humidity>Humidity: 38%</humidity>
        <wind>Wind: SE at 5 mph</wind>
        <pressure></pressure>
        <astronomy>
            <sunrise></sunrise>
            <sunset></sunset>
        </astronomy>
    </current_weather>
    <forecast_weather>
        <next_weather>
            <day>Tue</day>
            <temperature_max>82</temperature_max>
            <temperature_min>57</temperature_min>
            <condition>Clear</condition>
        </next_weather>
        <next_weather>
            <day>Wed</day>
            <temperature_max>80</temperature_max>
            <temperature_min>55</temperature_min>
            <condition>Chance of Rain</condition>
        </next_weather>
    </forecast_weather>
</weather>
```

Figura 6.8: Exemplo de resposta construído com base na abstracção concebida no mediador de serviços meteorológicos

No Apêndice B encontram-se extractos das respostas dos dois fornecedores a um pedido com o objectivo de obter as condições meteorológicas para uma dada localização. Com base na estruturação presente nestas respostas, concebeu-se uma estrutura própria, em formato `XML` tal como os originais, capaz de conter todos os elementos presentes nos formatos de resposta dos dois fornecedores. Na Figura 6.8 apresenta-se um exemplo de um documento `XML` no formato concebido.

O objectivo do módulo de abstracção de cada executor é mapear os resultados obtidos para a estrutura concebida por forma a proporcionar ao cliente a noção de transparência desejada. No entanto, devido aos formatos devolvidos pelo *GoogleWeather* e no *Yaho-*

oWeather serem diferentes, cada módulo de abstracção é específico de cada executor. Saliente-se que para além da abstracção relativa ao formato `XML`, o módulo de abstracção implementado possui métodos para fornecer a informação recolhida ao cliente sob um formato textual de forma a tornar fácil a leitura dos resultados obtidos.

6.5.2 Mediador para Mapas

O serviço de mapas pretende fornecer aos seus clientes a possibilidade de incluírem nas suas aplicações um mapa geográfico (ver Figura 6.9) em que possam navegar por forma a encontrar uma localização ou região, ou no caso dos programadores, associar determinados elementos (marcadores por exemplo) a locais por forma a transmitirem informações de uma forma geográfica.

Figura 6.9: Exemplo de mapa fornecido pelo *BingMaps*

No contexto do sistema de mediação, o caso relativo à mediação para o serviço de mapas é de certa forma diferente das restantes. Por norma, a implementação de um mapa em aplicações Web é efectuada através da utilização de código *JavaScript* que é embutido no código do cliente. Desta forma, a comunicação entre o servidor e o cliente é totalmente encapsulada, o que confere inúmeras vantagens face ao modo de invocação tradicional, como por exemplo, garantir ao cliente que este pode navegar no mapa sem que necessite de adicionar novo código ou implementar mecanismos para detecção de eventos no mapa (embora o possa fazer). No entanto, este modelo também impossibilita que a noção de middleware seja implementada da mesma forma que o mediador de meteorologia. Neste sentido e no âmbito do sistema de mediação, este exemplo pretende mostrar a capacidade de adaptação do modelo arquitectural proposto.

A utilização de serviços de mapas baseia-se na inclusão de código nas aplicações Web. Como tal, a solução encontrada assenta no mesmo princípio e compreende a devolução ao cliente, consoante as suas pretensões, de um bloco de código concebido para esse efeito. Este bloco, devido à diferença patente no código dos diversos fornecedores, será encapsulado numa estrutura abstracta por forma a, tal como acontece com os restantes mediadores, retirar do cliente a preocupação de qual o fornecedor que respondeu ao pedido. Desta forma, este processo permite ao cliente incluir dinamicamente (através de código *JavaScript*) um mapa na página Web onde o deseja colocar.

Análise ao Serviço

A implementação do serviço de mapas no sistema de mediação foi concebida com base em três fornecedores, o *GoogleMaps*, o *YahooMaps* e o *BingMaps*.

Estes três fornecedores possuem formas de implementação muito semelhantes. Por forma a serem utilizados, é necessário que o programador inclua na sua página Web um bloco de código composto por duas secções obrigatórias.

```
<script type="text/javascript" src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.2"></script>
<script type="text/javascript">
    function startMap(){

    var map = new VEMap("mymap");

    map.HideDashboard();
    map.ShowDashboard();

    var latitude =46;
    var longitude =-42;

    var startingLocation = new VELatLong(latitude,longitude);
    map.LoadMap(startingLocation, 13);
    }
</script>
```

Figura 6.10: Exemplo Minimalista de Implementação do *BingMaps* numa página Web

Ilustrado na Figura 6.10, encontra-se um bloco de código destinado à implementação do serviço de mapas da Microsoft, o BingMaps, numa página Web. Na sua composição é perceptível a presença de dois elementos *script*. Estes elementos destinam-se a propósitos diferentes que, através da sua junção, definirão o modo como o mapa será invocado e utilizado.

A navegação patente do mapa deriva principalmente do primeiro *script*. De facto, é o primeiro *script* quem estabelece uma ligação com o servidor do fornecedor, e quem processa todas as alterações efectuadas ao mapa durante a sua utilização. Com isto, o fornecedor pretende retirar do programador a preocupação de analisar cada acção executada no mapa e consequentemente decidir como a irá processar. Apesar disto, todo este manuseamento pode prejudicar o programador em casos em que este pretenda analisar determinados eventos ou incluir aspectos na interface do mapa.

Por forma a complementar aspectos relacionados com a lógica de negócio pretendida pelo programador, em adição ao *script* já descrito, um segundo *script* terá que ser definido, como visível na Figura 6.10. Neste bloco, o programador pode incluir aspectos que pretenda ver incluídos na utilização do mapa. De facto, por forma a inicializar o mapa, este bloco terá que conter (obrigatoriamente) determinadas instruções nesse sentido. Algumas destas instruções centram-se com a definição do bloco * * onde o mapa irá aparecer na página Web, o ponto geográfico onde o mapa se irá centrar aquando da sua invocação ou que tipo de mapa o programador pretende que seja composto. Em adição, outros aspectos inclusíveis podem corresponder a uma localização que o utilizador pretenda ver saliente no mapa ou por exemplo a inclusão de marcadores com determinada informação associadas a coordenadas geográficas, entre outras.

Apesar da similaridade (em conceitos) na implementação dos serviços, cada um dos fornecedores de serviço possui funcionalidades exclusivas que podem ser importantes para o cliente do sistema de mediação e que podem ser consideradas durante o processo de eleição do melhor fornecedor. Neste sentido, após uma análise aos serviços, é possível estabelecer a seguinte tabela baseada na relação entre a funcionalidade e o fornecedor:

Funcionalidade	**GoogleMaps**	**YahooMaps**	**BingMaps**
Informação de Trânsito	Sim	Sim	Sim
Planeamento de Rotas	Sim	Não	Sim
Pesquisa de Indivíduos	Não	Não	Sim
Pesquisa de Empresas	Sim	Sim	Sim
Visionamento 3D	Não	Não	Sim
Olho de Falcão	Não	Não	Sim
StreetView	Sim	Não	Não

Tabela 6.1: Resultado da análise de funcionalidades presentes nos fornecedores do serviço de mapas

Para além destas funcionalidades, existem outras, mas que foram omitidas da tabela por serem compartilhadas por todos estes fornecedores. Algumas destas funcionalidades são por exemplo, a capacidade de *zoom*, a capacidade de colocar marcadores no mapa, a capacidade de visionar a rede de estradas, entre outras. No entanto, por forma a contemplar a possível adição de novos fornecedores ao mediador, estas funcionalidades foram também incluídas na base de dados do sistema.

Construção do mediador

Apesar da diferença na forma como o mediador compõe os resultados do serviço de mapas em relação aos restantes módulos, o método de implementação do modelo do mediação apresentado no Capítulo 5 permanece essencialmente igual. Assim, à semelhança dos restantes mediadores, o mediador para mapas possui como elementos constituintes o *Controlador*, o *Analisador*, o *Best Provider*, o *Mediador Intermédio* e os *Executores*.

Resumidamente, o módulo *Controlador* estabelece a recepção dos pedidos provenientes da camada de comunicações através da implementação dos métodos disponíveis no mediador de mapas. O módulo *Analisador* separa os parâmetros presentes na lista de argumentos entregue pelo *resource* da camada de comunicações. O módulo *Best Provider* efectua a selecção do fornecedor a utilizar com base nos parâmetros de eleição de fornecedor entregues pelo analisador. Por fim, o *Mediador Intermédio* invoca o módulo *Executor* (de acordo com o método invocado) correspondente à selecção efectuada anteriormente.

A diferença fundamental na implementação deste mediador face ao descrito anteriormente atenta no módulo de execução, principalmente na abstracção de resultados. Como referido anteriormente, devido à forma de execução do serviço de mapas, a solução encontrada passa pela devolução de um bloco de código que o cliente possa utilizar na sua aplicação. Neste sentido, presentes no mediador, definiram-se três métodos por forma a caracterizar esta funcionalidade:

- *getMapScript()* — obtém um bloco de código destinado a embutir um mapa numa página Web.
- *getFullMapScript()* — devolve uma página *html* completa com um mapa embutido.
- *createMarker()* — devolve um marcador a ser introduzido num mapa.

Através destas operações, o cliente será capaz de incluir, em tempo de execução da sua aplicação, um mapa composto com base, por exemplo, nas pretensões do utilizador da aplicação. No entanto, como descrito, cada um destes métodos irá devolver código executável para o cliente. De facto, este formato de entrega origina de certa forma um problema de confiança("*trusting*") entre o cliente e o sistema de mediação. No entanto, no âmbito desta tese, assume-se que existe confiança do cliente relativamente aos fornecedores do sistema de mediação.

Composição dos resultados

À luz do modelo arquitectural proposto, o mediador de mapas possuí um *Executor* por cada fornecedor de serviço presente no sistema. A principal diferença centra-se com

o processo de execução destes módulos. De facto, contrariamente ao que sucede com outros mediadores, neste caso, o *Executor* não possui a função de invocar o servidor do fornecedor elegido, mas sim a de compor um bloco de código que posteriormente o possa fazer.

Neste sentido, é necessário antes de mais perceber o que pode conter este bloco de código. Anteriormente, foi descrito que a implementação de um fornecedor de mapas é efectuada através de código *JavaScript*, dividido em duas secções principais, uma relativa às comunicações com o servidor do fornecedor e a outra relativa a aspectos de implementação que o programador deseje incluir na operação do mapa. Como tal, o resultado da operação do mediador possuirá logicamente o mesmo formato.

No entanto, a devolução do código respectivo a cada fornecedor pode causar problemas de interacção ao programador se este quiser por exemplo incluir possíveis interacções entre outros elementos presentes na sua aplicação e o mapa. Isto sucede porque cada fornecedor pode implementar determinadas acções, como colocar um marcador ou definir o tipo de mapa, de formas diferentes.

Por forma a dar resposta a este problema, a solução encontrada passa por abstrair estas mesmas acções, ou seja, definir elementos que englobem determinadas instruções. Por exemplo, o processo de inicialização dos controladores do mapa tem diferentes implementações para os vários fornecedores, isto é, enquanto a implementação relativa ao *BingMaps* necessita apenas de invocar um método, o *YahooMaps* necessita de três. No entanto, estas instruções referem-se à mesma operação, o lançamento dos controlos do mapa. Com base neste princípio, foram definidas abstracções para grupos de instruções que possam representar determinadas acções à semelhança da anterior.

De forma a construir estas abstracções recorreu-se à linguagem *JavaScript*. Em *JavaScript*, a abstracção de um grupo de instruções pode ser facilmente conseguida através da utilização de funções. Deste modo, constituíram-se funções que designem determinadas acções de acordo com cada fornecedor. Por forma a demonstrar a construção da resposta a este mediador, será utilizado o método "*getMapScript()*".

Este método tem como objectivo a devolução de um bloco de código constituído por dois *scripts* que o cliente irá embutir na sua página Web por forma a obter um mapa. Enquanto que o primeiro *script* não necessita de qualquer abstracção visto tratar-se apenas da especificação do endereço que o mapa utiliza para estabelecer a ligação — de forma a proporcionar a navegação ao cliente — o segundo *script* depende do fornecedor eleito e como tal, necessitará de ser abstraído.

No contexto da invocação de um mapa, é necessário (independentemente do fornecedor) que a segunda secção do bloco de código possua certos elementos. Para todos

estes elementos foram constituídas abstracções. No caso deste método, foram constituídas abstracções, designadamente para a inicialização do mapa, inicialização dos controlos do mapa, constituição de um marcador, definição do tipo de mapa, entre outras. Um exemplo destas funções encontra-se ilustrado na Figura 6.11.

```
function startControls(map){
    map.addTypeControl();
    map.addZoomLong();
    map.addPanControl();
}

function FindCoord(map,latitude, longitude){
    var startlocation = new YGeoPoint(latitude,longitude);
    map.drawZoomAndCenter(startlocation, 4);
}

function Find(map,address){
    map.drawZoomAndCenter(address,4);
}

function Marker(map,latitude,longitude,descricao){
    var geoPoint=new YGeoPoint(latitude,longitude);
    var newMarker= new YMarker(geoPoint);
    newMarker.addAutoExpand(descricao);
    map.addOverlay(newMarker);
}
```

Figura 6.11: Exemplo de Funções de Abstracção relativas ao *YahooMaps*

Após a definição dos elementos de construção do bloco de código, resta elaborar o processo de construção deste mesmo bloco. Inicialmente, o módulo *Executor* necessita de, tal como nos restantes mediadores, obter os parâmetros indicados no pedido. Para o método pré-designado, a invocação de um mapa requer que a longitude/latitude do ponto central do mapa seja fornecida, bem como o tipo de mapa que o cliente pretende. Desta forma, na invocação do *Executor* são-lhe entregues estes parâmetros (de acordo com o método seleccionado) que terão de ser incluídos na construção do bloco de código. Dado que, de acordo com o fornecedor eleito, serão necessárias diferentes instruções (relativas ao código), torna-se necessário armazenar estas instruções de acordo com o fornecedor correspondente, de forma a que sejam facilmente acedidas durante o processo de construção da resposta. Assim, no contexto do mediador de mapas, o código relativo a cada fornecedor foi armazenado numa base de dados auxiliar ao sistema, relativa apenas ao mediador em causa. Através das indicações provenientes do cliente e da selecção do fornecedor, o *Executor* irá elaborar um bloco de código, que posteriormente é devolvido ao cliente, utilizando os elementos presentes na base de dados do mediador. No Apêndice C pode-se encontrar um exemplo de resposta ao método *getMapScript()*. Os restantes métodos incorporam formatos semelhantes ao demonstrado pelo que serão omitidos.

6.5.3 Mediadores para Fotografia e *Messaging*

Os restantes serviços, messaging e fotografia incorporam no sistema de mediação, possuem uma implementação semelhante ao caso do mediador de meteorologia. Uma das diferenças relaciona-se com o facto de não possuírem concorrência, ou seja, apenas possuem um fornecedor de serviço. No entanto, este facto apenas traduz que a implementação compreende a inclusão de um só módulo *Executor*. Os restantes módulos foram todos implementados utilizando os processos anteriormente descritos, pelo que a sua descrição será omitida. Saliente-se apenas que, em ambos os casos, antevendo a adição de fornecedores concorrentes, foram constituídas estruturas de abstracção (ver Apêndice D), relativas a formatos de devolução de fotografias e álbuns no serviço de fotos, e relativas a mensagens no caso do serviço de *messaging*. Em adição, parâmetros relativos às funcionalidades dos fornecedores implementados foram incluídos desde logo na base de dados do sistema.

Um aspecto relevante respectivo a estes serviços é que requerem, contrariamente aos restantes, a implementação de mecanismos de autenticação de forma a ser possível aceder aos seus conteúdos. No Capítulo 4 foram levantadas questões relativas à autenticação, tendo sido propostas algumas soluções passíveis de serem utilizadas. Consequentemente, foi indicado que não existe uma solução definitiva e que a escolha de como implementar os mecanismos de autenticação deve ser efectuada no momento da implementação do grupo de serviços. Como tal, no contexto dos mediadores de fotos e *messaging*, a solução utilizada compreende a passagem dos parâmetros para autenticação no corpo do pedido efectuado ao sistema. Esta selecção deveu-se meramente ao facto de se tratarem de mediadores compostos apenas por um fornecedor o que exclui problemas relacionados com a eleição do fornecedor.

Relativamente aos fornecedores envolvidos, o mediador relativo ao serviço de armazenamento de fotografias contempla as funcionalidades presentes no *Picasa*. Algumas das funcionalidades implementadas permitem aos clientes do sistema de mediação criarem novos álbuns, obterem informação relativa aos mesmos ou obter listagens das fotografias presentes nestes. Por sua vez, o mediador de *messaging* contempla as funcionalidades do *Twitter*. As funcionalidades implementadas compreendem o envio de mensagens para o serviço, a criação e recepção de mensagens privadas ou o envio de convites.

6.6 Caso de Estudo

De forma a serem testadas as funcionalidades presentes no sistema de mediação construído, decidiu-se conceber uma aplicação Web capaz de utilizar conteúdos provenientes

de todas as mediações que aquele disponibiliza. Como solução, aplicação concebida, denominada *MySocial* propõe-se como uma rede social possibilitando aos seus utilizadores funcionalidades tais como:

- acesso a álbuns e fotografias;
- serviço de *messaging* entre os utilizadores da rede;
- visualização de mapas;

Adicionalmente, para além destas funcionalidades, decidiu-se-se que existiria na aplicação uma conjunção entre alguns dos conteúdos provenientes do sistema. Por exemplo, decidiu-se que fossem fornecidas funcionalidades tais como sobrepor a localização de uma imagem sobre um mapa, ou fornecer ao utilizador as condições meteorológicas com base em coordenadas geográficas. Desta forma, a aplicação assume características típicas de um *Mashup* ao agrupar informação proveniente de diversos serviços, proporcionando a disposição da informação segundo novas perspectivas. De forma a constituir estes aspectos, utilizou-se, durante a elaboração da aplicação, fundamentos relativos ao método de construção de *Mashups* apresentado no Capítulo 2.

No sentido de implementar as referidas funcionalidades, a aplicação divide-se em duas partes, uma relativa à obtenção dos conteúdos (camada de negócio), e a outra relativa à elaboração da interface da aplicação (camada de apresentação).

6.6.1 Camada de Negócio da Aplicação/Obtenção dos Conteúdos

A concepção de uma rede social compreende uma lógica fundamentalmente assente sobre utilizadores e os seus contactos (*amigos*). Neste sentido, tornou-se necessário criar, paralelamente à obtenção dos conteúdos provenientes do sistema de mediação, mecanismos que reflictam este aspecto. Por forma a dar suporte à aplicação, foi criada uma base de dados com o objectivo de armazenar informação referente aos utilizadores. É com base nela que os mecanismos de autenticação da aplicação irão recair, assim como é nesta que se armazena a lista de contactos de cada utilizador.

Na Figura 6.12 encontra-se a estrutura geral referente à camada de negócio presente na aplicação. Como ilustrado, para além da informação referente aos utilizadores, é também responsabilidade da camada de negócio da aplicação obter os diversos conteúdos provenientes do sistema de mediação. Neste sentido, constituíram-se módulos específicos para cada um dos tipos de serviço incluídos na aplicação. De forma a ser descrito o processo de execução destes módulos, são apresentados de seguida os diversos passos utilizados nesse sentido, de acordo as fases presentes no método de construção de Mashups.

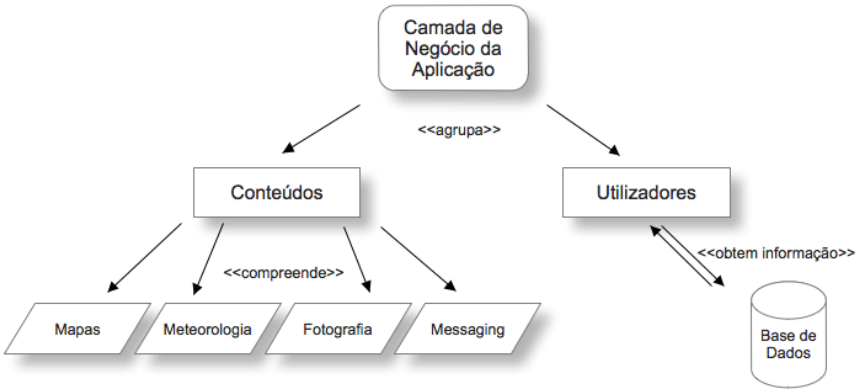

Figura 6.12: Ilustração dos diversos módulos presentes na camada de negócio da aplicação

Selecção e Agregação de Informação

O primeiro passo presente na elaboração de um Mashup tem como objectivo seleccionar as fontes de informação que serão utilizadas na aplicação. No caso em questão, este passo é bastante simplificado. Na verdade, dado que os diversos conteúdos a implementar são provenientes do sistema de mediação, não é necessária a procura por serviços capazes de fornecer os conteúdos pretendidos. No entanto, continua a ser necessária a escolha dos conteúdos a requerer de cada um destes serviços. De forma a compor a aplicação desejada, foram seleccionados dos diversos serviços as seguintes operações:

- Serviço de Fotografia — obtenção de álbuns, obtenção de fotografias e criação de álbuns;
- Serviço de Meteorologia — informação das condições meteorológicas com base numa localização;
- Serviço de *Messaging* — actualização do estado do cliente, obtenção da lista das mensagens do utilizador assim como a lista de mensagens enviadas por amigos, envio e recepção de mensagens privadas;
- Serviço de Mapas — obtenção de um mapa.

Por forma a obter estes conteúdos, em cada módulo relativo aos conteúdos, foram elaborados sub-módulos que, com base na `API` do sistema de mediação, interrogam, respectivamente, cada uma das operações pretendidas de forma a obterem os conteúdos desejados.

Organização e Classificação

Neste passo pretende-se que a informação recolhida seja organizada de acordo com o seu propósito de forma a ser facilmente acedida. No entanto, na implementação do passo anterior, isto já se encontra subjacente devido à construção de módulos capazes de obter a informação de acordo com o seu tipo de conteúdos. Neste sentido, este passo foi omitido.

Limpeza e Transformação dos Dados

De acordo com as funcionalidades pretendidas na aplicação, este passo pretende essencialmente processar a informação recolhida de forma a poder ser utilizada. Para tal, elaboraram-se na aplicação estruturas com o objectivo de apenas conter informação necessária para a execução da aplicação. Por exemplo, uma das funcionalidades pretendidas na aplicação consiste na visualização, por parte do utilizador, da lista de fotografias que este contém num dos seus álbuns. Neste sentido, decidiu-se que a interface a fornecer ao utilizador, irá apenas conter uma pequena representação da fotografia, o seu nome, data e descrição. No entanto, a resposta ao pedido efectuado ao sistema de mediação, nomeadamente ao serviço de fotografia, consiste num ficheiro XML com mais informação do que o necessário. Assim, por forma a ser seleccionada apenas a informação pretendida, construíram-se na aplicação módulos de *parsing* assentes sobre cada uma das operações da aplicação. Deste modo, será da responsabilidade de cada um destes módulos compor as estruturas pretendidas com a informação necessária para a interface da aplicação.

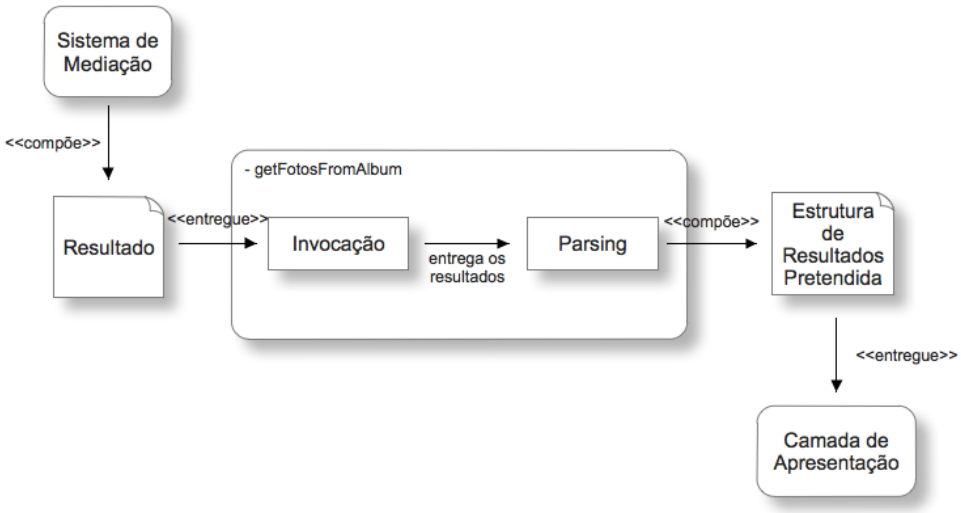

Figura 6.13: Ilustração do fluxo de dados de uma operação no contexto da camada de negócio da aplicação

Na Figura 6.13 é apresentado todo o processo associado à operação de obtenção de

álbuns (*getFotosFromAlbum*) contida num dos módulos implementados (fotografia). O seu processamento consiste na obtenção dos resultados provenientes do sistema de mediação, através da invocação do mesmo, seguindo-se a sua reestruturação para o formato esperado pela camada de apresentação. Note-se que todas as operações implementadas na aplicação se baseiam neste procedimento.

6.6.2 Camada de Apresentação/Interface da Aplicação Web

O último passo da constituição da aplicação retrata a implementação da camada de apresentação. A camada de apresentação tem como missão compor as interfaces da aplicação com base nos conteúdos fornecidos pela camada de negócio. No contexto apresentado, ou seja, a elaboração de uma rede social, foram apresentadas funcionalidades como o acesso a álbuns e fotografias, serviço de *messaging* e visualização de mapas em adição a funcionalidades derivadas da composição dos conteúdos existentes. Neste sentido, foi composta uma interface sub-dividida em três áreas: área de *messaging*, área relativa aos amigos e a área relativa aos álbuns e fotografias.

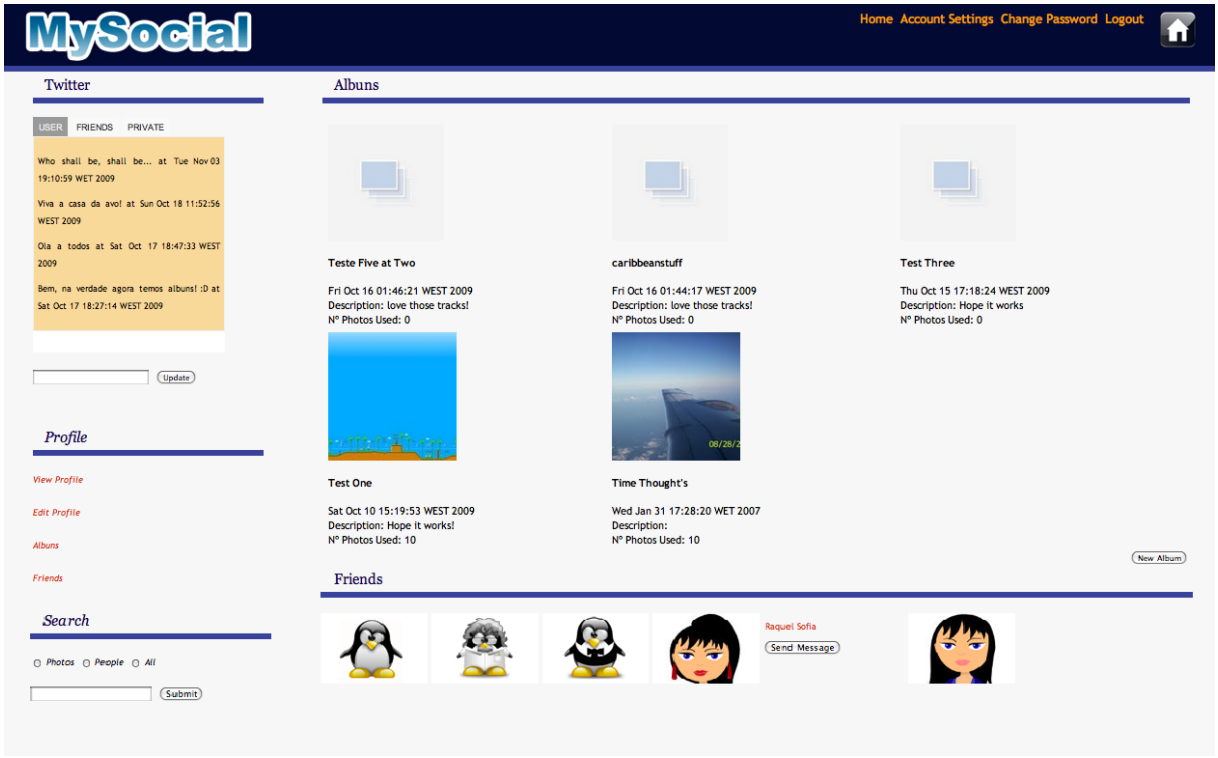

Figura 6.14: Apresentação da aplicação Web referente ao caso de estudo (MySocial)

A Figura 6.14 ilustra o aspecto geral da página referente aos aspectos descritos. A área relativa a *messaging* proporciona ao utilizador a actualização do seu estado, a obtenção das suas mensagens e a obtenção das mensagens a si enviadas pelos seus amigos. Por forma

a serem obtidas estas informações, é necessário que a aplicação conheça, na invocação do sistema de mediação, as credenciais de autenticação referentes ao serviço a utilizar. Dado tratar-se de uma aplicação exemplo, neste caso, foi estabelecido que o cliente no momento da inscrição na rede social, fornece à aplicação todas as credenciais necessárias para a autenticação junto dos fornecedores requisitados. Deste modo, após o efectuar do *login* do utilizador, são requisitados ao sistema de mediação os conteúdos necessários através da informação já presente na base de dados da aplicação.

Na zona inferior da interface encontra-se a área relativa aos amigos do utilizador. Note-se que a informação aqui disposta provém da própria lógica de negócio da aplicação e que, como tal, não depende dos conteúdos fornecidos pelo sistema de mediação. A única excepção, centra-se em permitir ao utilizador o envio de uma mensagem privada com base no serviço de *messaging* presente no sistema de mediação.

Por fim, na zona principal da página encontra-se a disposição dos conteúdos referentes aos álbuns e fotografias do utilizador. O processo de obtenção deste tipo de conteúdos é semelhante ao anterior. No entanto, em adição aos álbuns e fotografias, constitui-se também nesta área uma outra funcionalidade. Esta funcionalidade tem como objectivo fornecer ao utilizador a localização de uma das suas fotos anexada às condições meteorológicas, dispondo esta informação sobre um mapa. De facto, esta funcionalidade corresponde à fase final de construção de um Mashup, a visualização.

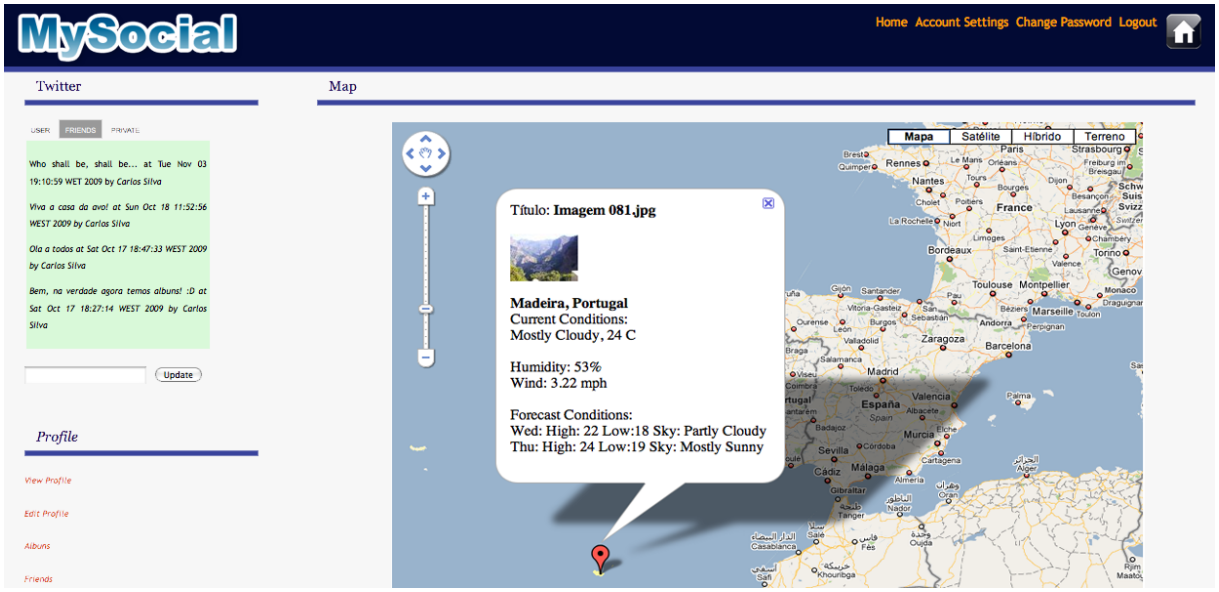

Figura 6.15: Aspecto do mapa na aplicação (MySocial) com a informação relativa à fotografia seleccionada

A implementação desta funcionalidade depreende que, aquando da selecção da fotografia por parte do utilizador, a aplicação, com base nas coordenadas geográficas e no

thumbnail da imagem, obtenha em primeiro lugar a informação meteorológica da região onde a foto foi criada e de seguida obtenha um mapa centrado sobre as mesmas coordenadas incorporando no mesmo um marcador com a imagem e as condições meteorológicas. O resultado desta operação é visível na Figura 6.15.

Apesar de todos os conteúdos necessários serem obtidos e estruturados pela camada de negócio da aplicação, a combinação e composição destes é apenas efectuada na camada de apresentação. Devido à utilização do sistema de mediação, a composição pode ser vista segundo uma perspectiva de serviços ao invés de fornecedores, dado que a junção de dois serviços pode representar várias combinações de fornecedores possíveis. Esta particularidade facilitou a concepção da aplicação, dado que, em adição a ser apenas necessário o estudo de uma `API` (a do sistema de mediação), foi apenas necessário compor os resultados já uniformizados provenientes dos mediadores do sistema.

Capítulo 7

Conclusões

A crescente utilização de serviços Web como fornecedores de conteúdos para a construção de aplicações é hoje uma realidade. O aparecimento de arquitecturas como SOA, focalizadas para a incorporação de serviços de forma remota, ou de modelos de entrega de software como SaaS, justificam este facto. No entanto, há ainda muitas dificuldades que se apresentam aos programadores de tais aplicações. O modelo arquitectural proposto nesta dissertação é um veículo de apoio ao desenvolvimento de aplicações Web que pretende resolver, ou pelo menos minorar, tais dificuldades.

Um dos principais problemas derivados da utilização de serviços Web provém da ocorrência de indisponibilidade. Na verdade, no contexto de uma aplicação Web, se um serviço que esta utilize se encontra indisponível, então toda a lógica de negócio associada à aplicação pode estar comprometida. Como consequência, a disponibilidade da própria aplicação será afectada, podendo mesmo ter influência na razão de confiança dos utilizadores relativamente à aplicação. De forma a reduzir a ocorrência deste problema, o modelo de mediação apresentado no Capítulo 5 assume-se como um mediador múltiplo capaz de reunir diversos fornecedores de serviço de acordo com o tipo de conteúdos que providenciam, de forma a garantir ao utilizador a existência de alternativas, na eventualidade de um dos fornecedores falhar. Com isto, a ocorrência de falhas de serviço de um fornecedor pode ser parcial ou totalmente eliminada, através da disponibilização do serviço de um fornecedor alternativo assente sobre a mesma lógica de negócio.

Durante a concepção de aplicações Web, a escolha dos fornecedores a incorporar é uma decisão difícil para o programador e para o stakeholder da aplicação. Neste sentido, aspectos relacionados com as funcionalidades do fornecedor ou aspectos relacionados com a qualidade do serviço podem influenciar a decisão a tomar. No entanto, alguns destes elementos são por vezes difíceis de obter e podem sofrer oscilações nos resultados, o que complica o processo efectivo de análise. Por forma a auxiliar os clientes nesta escolha, na

lógica do modelo proposto, foram constituídas políticas de melhor fornecedor utilizando padrões de qualidade de serviço (QoS), capazes de obter dados sobre o estado do serviço e estabelecer as suas condições gerais de funcionamento. Desta forma, o cliente não necessita de estudar os fornecedores, bastando-lhe indicar o critério de selecção do melhor fornecedor aquando do seu pedido.

Ao sistema de mediação compete realizar a monitorização constante aos serviços, quer a nível das suas funcionalidades, quer a nível da sua execução de forma a garantir que o fornecedor seleccionado é aquele que melhor prestará o serviço desejado. Adicionalmente, se na perspectiva do cliente existirem aspectos que este entenda como indispensáveis no serviço a requisitar, o modelo permite que estes sejam introduzidos no momento de invocação do sistema, sendo estes aspectos incorporados no processo de decisão do fornecedor a utilizar.

A construção de uma aplicação Web pode, por vezes, requerer a utilização de um elevado número de serviços Web em simultâneo. Neste caso, o programador tem que analisar cada API, e consequentemente implementar um método de invocação do serviço na aplicação. Este processo pode implicar um elevado dispêndio de tempo, porque as APIs dos vários fornecedores podem ser bastante diferentes. O modelo arquitectural, ao propor a existência de uma única API, segundo as linhas conceptuais presentes em SaaS, que engloba todas as APIs dos serviços associados pelo sistema, apresenta-se como um elemento facilitador do trabalho dos projectistas de aplicações Web. O programador, para além de não necessitar de implementar cada serviço consoante a tecnologia especificada, necessita apenas de perceber o que espera o serviço e o que lhe será devolvido. Desta forma, o modelo pretende que o cliente apenas se preocupe com os conteúdos que deseja ver devolvidos, sem que necessite de se preocupar em perceber qual dos fornecedores compôs a resposta. Para que isto seja possível, é necessário que exista uma uniformização de resultados que dependam do mesmo tipo de serviço. Por isso, o modelo especifica que os resultados de todos os fornecedores que forneçam o mesmo tipo de serviço devem passar por um processo de abstracção de forma a que os resultados obtidos pelo cliente sejam completamente transparentes em relação ao fornecedor utilizado.

Assente em quatro tipos de serviços distintos, foi desenvolvido um sistema de mediação, projectado de acordo com o modelo arquitectural proposto nesta dissertação. A implementação desse sistema mostrou que o modelo é válido, embora também tenha alertado para o facto de o acesso a serviços com autenticação precisar de um estudo mais aprofundada. Na secção 7.1 voltar-se-á a este assunto.

No sistema desenvolvido, foram incorporados três fornecedores de mapas, dois for-

necedores de serviços de meteorologia, um fornecedor de fotografias e um fornecedor de *messaging*. Os pedidos ao serviço de mapas são efectuados sempre da mesma forma, independentemente do fornecedor. O mesmo acontece com os serviços de meteorologia. Esta implementação reflectiu a validade do modelo na medida em que a obtenção de conteúdos de diversos fornecedores, agrupados pelo seu tipo de serviço, requer ao cliente a utilização de um só modo de invocação, sucedendo o mesmo com a obtenção dos resultados, devido às componentes de abstracção constituídas sobre todas as operações suportadas. De forma a estabelecer critérios de selecção para o cliente, a `API` do sistema foi constituída de maneira a reflectir aspectos importantes na sua óptica, cabendo ao programador apenas o estudo dos parâmetros de serviço já recolhidos e consequentemente indicá-los no momento de invocação do serviço. Adicionalmente, a construção do sistema através da introdução de quatro tipos de serviços provou a modularidade do modelo, estabelecendo a capacidade deste em facilmente incluir novos serviços e novos fornecedores.

Por último, foi desenvolvida uma aplicação Web, que usou o sistema de mediação referido anteriormente como único fornecedor de serviços. Esta aplicação, concebida através das ideias conceptuais de *mashup*, assenta sobre a lógica de um sistema de suporte às comunidades, proporcionando aos seus utilizadores o acesso a álbuns e fotografias, assim como a ferramentas de messaging. Através da utilização do sistema, a construção desta aplicação foi simplificada. Sem a existência do sistema de mediação, se se pretendesse conceber a mesma aplicação, seria necessário incorporar na própria aplicação as lógicas de negócio de cada fornecedor, construir abstracções para cada tipo de resultado, implementar cada uma das `API`s e constituir mecanismos de análise e eleição a cada serviço. Através do mediador, todos estes passos são retirados ao programador permitindo que este se concentre apenas na concepção da lógica de negócio da aplicação.

Em síntese, o modelo concebido pretende facilitar o desenvolvimento de aplicações baseadas em recursos dispersos pela Web. Através de aspectos como modularidade e expansibilidade, o modelo constitui-se como uma solução de mediação para programadores de aplicações Web.

7.1 Trabalho Futuro

Nesta secção apresentam-se duas direcções de trabalho futuro que se revelaram durante a execução do trabalho. Uma está associada ao acesso a fornecedores que exijam credenciais de autorização, como por exemplo os serviços de fotografia e *messaging*. A outra, está relacionada com o fornecimento de serviços de valor acrescentado que resultam da composição de outros serviços.

Autenticação

Os fornecedores de serviços de fotografia e messaging usados na construção do sistema de mediação descrito no Capítulo 6 apenas respondem a utilizadores registados, exigindo por isso credenciais na altura de satisfazer um pedido de serviço. É habitual a exigência de credenciais neste tipo de serviços. Por forma a obter resultados em tempo útil, optou-se por usar o mecanismo mais fácil de autenticação: envio de credenciais com login e password. Mas este mecanismo é também o mais inseguro, porque as credenciais circulam na rede em texto não codificado. Ao fazer circular na rede as credenciais de forma não cifrada está-se a expô-las a possíveis intercepções por terceiros, criando claramente uma situação de falta de segurança. Por isso, é necessário estudar a possibilidade de implementação de mecanismos de autenticação mais seguros. Este aspecto no entanto, não fazia parte dos objectivos deste trabalho.

Considerando o caso em que o cliente especifica o fornecedor a consultar, deve-se permitir a utilização de mecanismos de autenticação mais fortes, desde que os fornecedores os suportem. Por exemplo, o Picasa, fornecedor de serviços de fotografia, suporta tokens OAuth [Wor]. Este protocolo permite que aplicações e websites (consumidores) acedam a recursos protegidos de um web service (fornecedores) através de uma `API`, sem obrigar que os utilizadores divulguem as suas credenciais de serviço aos consumidores. É necessário estudar a forma de usar este tipo de autenticação quando um intermediário, o sistema de mediação, se interpõe entre o cliente e o fornecedor.

Mas, o problema da autenticação é mais vasto, porque deve ser pensado num contexto multi-fornecedor. A procura de uma solução passa pela obtenção de uma resposta à questão: Como lidar com as credenciais de acesso a um fornecedor que pode ainda não ter sido escolhido? Desejavelmente, a autenticação deveria ser estabelecida directamente entre o cliente e o fornecedor, não passando pelo sistema de mediação.

Composição de Serviços

Na consulta por dados meteorológicas no serviço da Yahoo (*YahooWeather*) a indicação do local é feito usando um código. Normalmente o cliente vê-se na necessidade de traduzir o local, do qual pretende ver as condições climatéricas, para o código que lhe corresponde de forma a poder consultar o serviço. Isto obriga-o a efectuar um pedido a outro serviço por forma a fazer esta mesma tradução. Usando o sistema de mediação desenvolvido é possível consultar o serviço meteorológico da Yahoo fornecendo o local, porque fica à responsabilidade do sistema fazer a tradução. Tal acontece porque o sistema de mediação faz internamente a composição de dois serviços no sentido de fornecer o seu. Este tipo de

procedimento, composição de serviços, pode ser explorado no sentido de criar serviços de valor acrescentado.

Bibliografia

[Akk07] G. B. Akkus. Semantic web services composition: A network analysis approach. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 937–943, 2007.

[BMS+08] Jack K. Beaton, Brad A. Myers, Jeffrey Stylos, Sae, and Yingyu (. Xie. Usability evaluation for enterprise soa apis. In *SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments*, pages 29–34, New York, NY, USA, 2008. ACM.

[Bor] Stephane Bortzmeyer. Echoping home page. `http://echoping.sourceforge.net/`.

[BSLZ08] Anders A. Bjerkestrand, Lars A. Skaar, Ruth Lennon, and Amir Zeid. Sixth international workshop on soa & web services: best practices. In *OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 857–858, New York, NY, USA, 2008. ACM.

[Cer02] Ethan Cerami. *Web Services Essentials.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[Cho07] Vidyanand Choudhary. Software as a service: Implications for investment in software development. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, page 209a, 2007.

[CLND09] Allen Cypher, Tessa Lau, Jeffrey Nichols, and Mira Dontcheva. Workshop on end user programming for the web. In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 4779–4782, New York, NY, USA, 2009. ACM.

[CNW01] Francisco Curbera, William A. Nagy, and Sanjiva Weerawarana. Web services: Why and how. In *In OOPSLA 2001 Workshop on Object-Oriented Web Services. ACM*, 2001.

[Cora] Microsoft Corporation. Com: Component object model technologies. `http://www.microsoft.com/com/default.mspx`.

[Corb] Microsoft Corporation. Home : The official microsoft silverlight site. `http://silverlight.net/`.

[Corc] Microsoft Corporation. Web services protocol stack. `http://msdn.microsoft.com/en-us/library/aa966274(BTS.10).aspx`.

[CW03] Sandeep Chatterjee and James Webber. *Developing enterprise web services: an architect's guide*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2003.

[DB07] Andrea D'Ambrogio and Paolo Bocciarelli. A model-driven approach to describe and predict the performance of composite services. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 78–89, New York, NY, USA, 2007. ACM Press.

[DS05] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *IJWGS*, 1(1):1–30, 2005.

[DVV03] Christos Doulkeridis, Efstratios Valavanis, and Michalis Vazirgiannis. Towards a context-aware service directory. pages 54–65. 2003.

[ECM08] Javier Espadas, David Concha, and Arturo Molina. Application development over software-as-a-service platforms. In *ICSEA '08: Proceedings of the 2008 The Third International Conference on Software Engineering Advances*, pages 97–104, Washington, DC, USA, 2008. IEEE Computer Society.

[EG07] Robert J. Ennals and Minos N. Garofalakis. Mashmaker: mashups for the masses. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1116–1118, New York, NY, USA, 2007. ACM.

[Fei05] E. Feig. Five years of software as a service: the good, the bad and the ugly. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages xxxiii vol.1+, 2005.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[Gar] Jesse Garret. ajax: a new approach to web applications. `http://adaptivepath.com/ideas/essays/archives/000385.php`.

[GG98] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[GL08] Qing Gu and Patricia Lago. Soa process decisions: new challenges in architectural knowledge modeling. In *SHARK '08: Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*, pages 3–10, New York, NY, USA, 2008. ACM.

[Gooa] Google. Google app engine - google code. `http://code.google.com/appengine/`.

[Goob] Google. Google mashup editor - google code. `http://code.google.com/gme/`.

[Groa] Object Management Group. Object management group - uml. `http://www.uml.org/`.

[Grob] Object Management Group. Omg's corba website. `http://www.corba.org/`.

[Gur04] Anura Gurugé. *Web Services: Theory and Practice*. Digital Press, 2004.

[Hat] Red Hat. Jboss - global leader in open source middleware software. `http://www.jboss.com`.

[HB08] Michael Hafner and Ruth Breu. *Security Engineering for Service-oriented Architectures*. Springer, October 2008.

[IBMa] IBM. Ibm - quick and easily done wiki. `http://services.alphaworks.ibm.com/graduated/qedwiki.html`.

[IBMb] IBM. Ibm software - websphere. `http://www-01.ibm.com/software/websphere`.

[IBMc] IBM. Understand enterprise service bus scenarios and solutions in service-oriented architecture, part 1. `https://www.ibm.com/developerworks/webservices/library/ws-esbscen/`.

[IHJ+07] Mamdouh H. Ibrhaim, Kerrie Holley, Nicolai M. Josuttis, Brenda Michelson, Dave Thomas, and John Devadoss. The future of soa: what worked, what didn't, and where is it going from here? In *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion*, pages 1034–1038, New York, NY, USA, 2007. ACM.

[Int] Intel. Intel mash maker. `http://mashmaker.intel.com`.

[JSO] JSON. Javascript object notation. `http://www.json.org/`.

[KFS+06] A. Kozlenkov, V. Fasoulas, F. Sanchez, G. Spanoudakis, and A. Zisman. A framework for architecture-driven service discovery. In *SOSE '06: Proceedings of the 2006 international workshop on Service-oriented software engineering*, pages 67–73, New York, NY, USA, 2006. ACM.

[KLZ02] Shonali Krishnaswamy, Seng Wai Loke, and Arkady Zaslavsky. Application run time estimation: a quality of service metric for web-based data mining services. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 1153–1159, New York, NY, USA, 2002. ACM.

[LSRH08] D. Lizcano, J. Soriano, M. Reyes, and J. J. Hierro. Ezweb/fast: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming “ubiquitous soa”. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM '08. The Second International Conference on*, pages 488–495, 2008.

[LT08] Hancheng Liao and Changqi Tao. An anatomy to saas business mode based on internet. In *Management of e-Commerce and e-Government, 2008. ICMECG '08. International Conference on*, pages 215–220, 2008.

[LZV08] Phillip A. Laplante, Jia Zhang, and Jeffrey Voas. What's in a name? distinguishing between saas and soa. *IT Professional*, 10(3):46–50, 2008.

[Mica] Microsoft. Microsoft popfly. `http://www.popfly.com`.

[Micb] Sun Microsystems. Java ee at a glance. `http://java.sun.com/javaee/`.

[MMD06] Sudarshan Murthy, David Maier, and Lois Delcambre. Mash-o-matic. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, pages 205–214, New York, NY, USA, 2006. ACM.

[Moz] Mozilla. Mozilla labs » ubiquity. `http://labs.mozilla.com/projects/ubiquity/`.

[MRT07] Michael E. Maximilien, Ajith Ranabahu, and Stefan Tai. Swashup: situational web applications mashups. In *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion*, pages 797–798, New York, NY, USA, 2007. ACM.

[MS08] S. Kami Makki and Jaina Sangtani. Data mashups & their applications in enterprises. In *ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pages 445–450, Washington, DC, USA, 2008. IEEE Computer Society.

[Nat07] Yefim Natis. Introducing saas-enabled application platforms: Features, roles and futures. Gartner Inc., 2007.

[Nes08] Tobias Nestler. Towards a mashup-driven end-user programming of soa-based applications. In *iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 551–554, New York, NY, USA, 2008. ACM.

[New02] Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley Professional, May 2002.

[OASa] OASIS. Oasis web services business process execution language (wsbpel) tc. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.

[OASb] OASIS. Uddi | online community for universal description, discovery and integration. `http://uddi.xml.org/`.

[OBG08] Liam O'Brien, Paul Brebner, and Jon Gray. Business transformation to soa: aspects of the migration and performance and qos issues. In *SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments*, pages 35–40, New York, NY, USA, 2008. ACM.

[OMB07] Liam O'Brien, Paulo Merson, and Len Bass. Quality attributes for service-oriented architectures. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, Washington, DC, USA, 2007. IEEE Computer Society.

[PH07] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, July 2007.

[PZL08] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 805–814, New York, NY, USA, 2008. ACM.

[Ran03] Shuping Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1):1–10, 2003.

[RHC08] Muhammad Raza, Farookh khadeer Hussain, and Elizabeth Chang. A methodology for quality-based mashup of data sources. In *iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 528–533, New York, NY, USA, 2008. ACM.

[RR07] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, Inc., May 2007.

[Shi06] Xuan Shi. Sharing service semantics using soap-based and rest web services. *IT Professional*, 8(2):18–24, 2006.

[SMK+03] Stanley Y. W. Su, Jie Meng, Raja Krithivasan, Seema Degwekar, and Sumi Helal. Dynamic inter-enterprise workflow management in a constraint-based e-service infrastructure. *Electronic Commerce Research*, 3(1-2):9–24, 2003.

[SML08] Thorsten Scheibler, Ralph Mietzner, and Frank Leymann. Eai as a service - combining the power of executable eai patterns and saas. *Enterprise Distributed Object Computing Conference, IEEE International*, 0:107–116, 2008.

[Stea] M. Stevens. The benefits of a service-oriented architecture. `http://www.developer.com/services/article.php/1041191`.

[Steb] M. Stevens. Service-oriented architecture introduction, part 1. `http://www.developer.com/services/article.php/10928_1010451_2`.

[Sun] Sun. Java message service - jms. `http://java.sun.com/products/jms/`.

[SW08] Ricky E. Sward and Kelly J. Whitacre. A multi-language service-oriented architecture using an enterprise service bus. *Ada Lett.*, 28(3):85–90, 2008.

[Tak06] Len Takeuchi. Asps: The integration challenge. *Queue*, 4(5):46–52, 2006.

[TBB03] M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *Computer*, 36(10):38–44, 2003.

[Tec] Noelios Technologies. Restlet - restful web framework for java. `http://www.restlet.org/`.

[TFC+06] W. T. Tsai, Chun Fan, Yinong Chen, Raymond Paul, and Jen-Yao Chung. Architecture classification for soa-based applications. In *ISORC '06: Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 295–302, Washington, DC, USA, 2006. IEEE Computer Society.

[TGWC06] W. T. Tsai, Jerry Gao, Xiao Wei, and Yinong Chen. Testability of software in service-oriented architecture. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference*, pages 163–170, Washington, DC, USA, 2006. IEEE Computer Society.

[TV08] G. Thies and G. Vossen. Web-oriented architectures: On the impact of web 2.0 on service-oriented architectures. In *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pages 1075–1082, 2008.

[US] Inc. UserLand Software. Xml-rpc home page. `http://www.xmlrpc.com/`.

[W3Ca] W3C. Html 4.01 specification. `http://www.w3.org/TR/html4/`.

[W3Cb] W3C. Http: Hypertext transfer protocol overview. `http://www.w3.org/Protocols/`.

[W3Cc] W3C. Simple object protocol language (soap). `http://www.w3.org/TR/soap`.

[W3Cd] W3C. Web service definition language (wsdl). `http://www.w3.org/TR/wsdl`.

[W3Ce] W3C. Web services glossary. `http://www.w3.org/TR/ws-gloss/`.

[W3Cf] W3C. World wide web consortium - web standards. `http://www.w3.org`.

[W3S] W3Schools. Dhtml introduction. `http://www.w3schools.com/dhtml/dhtml_intro.asp`.

[Wat07] Stephen Watt. Mashups - the evolution of the soa: Part 2: Situational applications and the mashup ecosystem. `http://www.ibm.com/developerworks/webservices/library/ws-soa-mashups2/`, 2007.

[Wor] OAuth Core Workgroup. Oauth core 1.0. `http://oauth.net/core/1.0a`.

[Yah] Yahoo. Pipes: Rewire the web. `http://pipes.yahoo.com/pipes/`.

[YBCD08] Jin Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, 2008.

[Zan08] Nan Zang. Mashups: Who? what? why? 2008.

[ZR08] Nan Zang and Mary B. Rosson. What's in a mashup? and why? studying the perceptions of web-active end users. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 31–38, 2008.

Apêndice A

API do Sistema

De seguida são apresentados as operações implementadas pelo sistema de mediação de acordo com o tipo de serviço a que pertencem. Salienta-se que de forma a simplificar a sua descrição, valores referentes a parâmetros de monitorização não serão incluídos em cada método (apenas no primeiro) dado serem constantes em todos as operações. As operações orientadas para um fornecedor em particular são também omitidas. A sua estrutura difere por conter apenas os parâmetros do método e a indicação do fornecedor no `URL` como demonstrado na Secção 6.2.

Mapas

- *Operação* — createMap
- *Tipo* — GET
- *Parâmetros Método* :
  localizacao – *endereço do local onde o mapa se centra*;
  latitude/longitude – *coordenadas geográficas onde o mapa se centra*;
  tipomapa – *específica se o mapa terá vista normal, por satélite ou híbrida*;
- *Parâmetros Funcionalidades de Serviço* :
  traffic – *inclusão de informação sobre o trânsito (true/false)*;
  routeplanner – *capacidade de planeamento de rotas (true/false)*;
  businessfinder – *capacidade de pesquisa de empresas (true/false)*;
  streetview – *capacidade de pesquisa de empresas (true/false)*);
  birdseye – *capacidade de pesquisa de empresas (true/false)*);
  peoplefinder – *capacidade de pesquisa de empresas (true/false)*);
- *Parâmetros Monitorização* :
  disponibilidade – *a escolha do fornecedor contempla a disponibilidade (true/false)*
  desempenho – *a escolha do fornecedor contempla o desempenho (true/false)*
  latencia – *a eleição do fornecedor contempla a latência (true/false)*
  responsetime – *a eleição do fornecedor contempla os tempos de resposta (true/false)*));
- *URL* — http://medsys/maps/createMap

- *Resposta* — 200 OK & Texto (*Duas tags script contendo o mapa*);
  404 Erro
- *Descrição* — Devolve código JavaScript correspondente a um mapa. Por forma a construir o mapa, o utilizador deve invocar a operação "startMap()" presente no código devolvido.

- *Operação* — createFullMap
- *Tipo* — GET
- *Parâmetros Método* :
  localizacao – *endereço do local onde o mapa se centra*;
  latitude/longitude – *coordenadas geográficas onde o mapa se centra*;
  tipomapa – *específica se o mapa terá vista normal, por satélite ou híbrida*;
  comprimento – *específica o comprimento do mapa incluído*;
  altura – *específica a altura do mapa incluído*;
- *Parâmetros Funcionalidades de Serviço* :
  traffic – *inclusão de informação sobre o trânsito (true/false)*;
  routeplanner – *capacidade de planeamento de rotas (true/false)*;
  businessfinder – *capacidade de pesquisa de empresas (true/false)*;
  streetview – *capacidade de pesquisa de empresas (true/false)*);
  birdseye – *capacidade de pesquisa de empresas (true/false)*);
  peoplefinder – *capacidade de pesquisa de empresas (true/false)*);
- *URL* — http://medsys/maps/createMarker
- *Resposta* — 200 OK & Texto (*Um documento HTML*);
  404 Erro
- *Descrição* — Devolve uma página HTML com um mapa no seu interior.

- *Operação* — createMarker
- *Tipo* — GET
- *Parâmetros Método* :
  localizacao – *endereço do local onde o marcador se centra*;
  latitude/longitude – *coordenadas geográficas onde o marcador se centra*;
  descricao – *específica o conteúdo do marcador*;
- *Parâmetros Funcionalidades de Serviço* :
  traffic – *inclusão de informação sobre o trânsito (true/false)*;
  routeplanner – *capacidade de planeamento de rotas (true/false)*;
  businessfinder – *capacidade de pesquisa de empresas (true/false)*;
  streetview – *capacidade de visionar as ruas in loco(true/false)*);
  birdseye – *capacidade de visionamento 3D (true/false)*);
  peoplefinder – *capacidade de pesquisa de indivíduos (true/false)*);
- *URL* — http://medsys/maps/createMarker
- *Resposta* — 200 OK & Texto (*Um documento HTML*);
  404 Erro
- *Descrição* — Devolve um marcador concebido de forma a ser embutido no mapa devolvido pela função *createMap*.

Meteorologia

- *Operação* — obtainMeteoDataText
- *Tipo* — GET
- *Parâmetros Método* :
  localizacao – *local de onde se pretende obter as condições meteorológicas (Local,Pais)*;
  unidadetemperatura – *especifica a unidade de temperatura (Celsius/Fahrenheit)*;
- *Parâmetros Funcionalidades de Serviço* :
  pressao – *inclusão de informação relativa à pressão atmosférica (true/false)*;
  suntimes – *requisição de informação relativa às horas de levantar/pôr do sol (true/false)*;
  nextdayprevisions – *requisição de informação relativa aos próximos dias (true/false)*;
  velocidadevento – *requisição da velocidade do vento (true/false)*;
  humidade – *inclusão da humidade relativa (true/false)*;
  mintemperature – *inclusão da temperatura mínima relativa ao local solicitado (true/false)*;
  maxtemperature – *inclusão da temperatura máxima relativa ao local solicitado (true/false)*;
- *URL* — http://medsys/weather/obtainMeteoDataText
- *Resposta* — 200 OK & Text (*Um parágrafo em HTML*);
  404 Erro
- *Descrição* — Devolve um parágrafo em HTML contendo toda a informação requisitada de forma a ser facilmente utilizada pelo cliente.

- *Operação* — obtainMeteoDataXML
- *Tipo* — GET
- *Parâmetros Método* :
  localizacao – *local de onde se pretende obter as condições meteorológicas (Local,Pais)*;
  unidadetemperatura – *especifica a unidade de temperatura (Celsius/Fahrenheit)*;
- *Parâmetros Funcionalidades de Serviço* :
  pressao – *inclusão de informação relativa à pressão atmosférica (true/false)*;
  suntimes – *requisição de informação relativa às horas de levantar/pôr do sol (true/false)*;
  nextdayprevisions – *requisição de informação relativa aos próximos dias (true/false)*;
  velocidadevento – *requisição da velocidade do vento (true/false)*;
  humidade – *inclusão da humidade relativa (true/false)*;
  mintemperature – *inclusão da temperatura mínima relativa ao local solicitado (true/false)*;
  maxtemperature – *inclusão da temperatura máxima relativa ao local solicitado (true/false)*;
- *URL* — http://medsys/weather/obtainMeteoDataXML
- *Resposta* — 200 OK & XML;
  404 Erro
- *Descrição* — Devolve um documento XML contendo toda a informação requisitada segundo uma estrutura pré-definida.

Fotografia

- *Operação* — createAlbum
- *Tipo* — POST
- *Parâmetros Método* :
  username – *credencial referente ao nome do utilizador*;
  password – *credencial referente à palavra passe do utilizador*;
  albumname – *nome a atribuir ao álbum*;
  descricao – *descrição do álbum a criar*;
- *URL* — http://medsys/photos/createAlbum
- *Resposta* — 200 OK;
  404 Erro
- *Descrição* — Cria um álbum no serviço de fotografia Picasa.

- *Operação* — checkAlbunsContents
- *Tipo* — GET
- *Parâmetros Método* :
  username – *credencial referente ao nome do utilizador*;
  password – *credencial referente à palavra passe do utilizador*;
- *URL* — http://medsys/photos/checkAlbunsContents
- *Resposta* — 200 OK & XML;
  404 Erro
- *Descrição* — Obtém a lista dos álbuns presentes no Picasa, referentes ao utilizador
  introduzido, estruturado num documento `XML`.

- *Operação* — getAllPhotosFromAnAlbum
- *Tipo* — GET
- *Parâmetros Método* :
  username – *credencial referente ao nome do utilizador*;
  password – *credencial referente à palavra passe do utilizador*;
  albumid – *identificador do álbum solicitado*
- *URL* — http://medsys/photos/getAllPhotosFromAnAlbum
- *Resposta* — 200 OK & XML;
  404 Erro
- *Descrição* — Obtém a lista de fotografias presentes num álbum especificado, referentes ao utilizador
  introduzido, estruturado num documento `XML`.

- *Operação* — getPublicPhotosBySearch
- *Tipo* — GET
- *Parâmetros Método* :
  searchterm – *expressão de procura por fotografias*;
- *URL* — http://medsys/photos/getPublicPhotosBySearch
- *Resposta* — 200 OK & XML;
  404 Erro
- *Descrição* — Obtém uma lista de fotografias de domínio público de acordo com
  a expressão introduzida, estruturada num documento `XML`.

Messaging

- *Operação* — updateStatus
- *Tipo* — POST
- *Parâmetros Método* :
  senderID – *credencial referente ao nome do utilizador no serviço*;
  password – *credencial referente à palavra passe do utilizador*;
  message – *mensagem a enviar*
- *URL* — http://medsys/messaging/updateStatus
- *Resposta* — 200 OK;
  404 Erro
- *Descrição* — Actualiza o estado do utilizador no Twitter com uma nova mensagem.

- *Operação* — getFriendsMessages
- *Tipo* — GET
- *Parâmetros Método* :
  senderID – *credencial referente ao nome do utilizador no serviço*;
  password – *credencial referente à palavra passe do utilizador*;
- *URL* — http://medsys/messaging/getFriendsMessages
- *Resposta* — 200 OK & XML;
  404 Erro
- *Descrição* — Obtém uma lista de mensagens enviadas pelos amigos do utilizador no Twitter, estruturada num documento `XML`.

- *Operação* — getUserMessages
- *Tipo* — GET
- *Parâmetros Método* :
  senderID – *credencial referente ao nome do utilizador no serviço*;
  password – *credencial referente à palavra passe do utilizador*;
- *URL* — http://medsys/messaging/getUserMessages
- *Resposta* — 200 OK & XML;
  404 Erro
- *Descrição* — Obtém a lista de mensagens enviadas pelo utilizador no Twitter, estruturada num documento `XML`.

- *Operação* — sendPrivateMessage
- *Tipo* — POST
- *Parâmetros Método* :
  senderID – *credencial referente ao nome do utilizador no serviço*;
  password – *credencial referente à palavra passe do utilizador*;
  receiverID – *credencial referente ao destinatário da mensagem*;
  message – *mensagem a enviar*;
- *URL* — http://medsys/messaging/sendPrivateMessage
- *Resposta* — 200 OK;
  404 Erro
- *Descrição* — Envia uma mensagem particular ao destinatário especificado.

- *Operação* — receivePrivateMessages
- *Tipo* — GET
- *Parâmetros Método* :
  senderID – *credencial referente ao nome do utilizador no serviço*;
  password – *credencial referente à palavra passe do utilizador*;
- *URL* — http://medsys/messaging/receivePrivateMessages
- *Resposta* — 200 OK & XML;
  404 Erro
- *Descrição* — Obtém a lista de mensagens privadas do utilizador, estruturadas num documento `XML`.

- *Operação* — validateAuthentication
- *Tipo* — GET
- *Parâmetros Método* :
  senderID – *credencial referente ao nome do utilizador no serviço*;
  password – *credencial referente à palavra passe do utilizador*;
- *URL* — http://medsys/messaging/validateAuthentication
- *Resposta* — 200 OK;
  404 Erro
- *Descrição* — Efectua o processo de autenticação perante o serviço Twitter.

- *Operação* — createFriendship
- *Tipo* — GET
- *Parâmetros Método* :
  senderID – *credencial referente ao nome do utilizador no serviço*;
  password – *credencial referente à palavra passe do utilizador*;
  receiveID – *credencial referente ao utilizador que se pretende adicionar*;
- *URL* — http://medsys/messaging/createFriendship
- *Resposta* — 200 OK;
  404 Erro
- *Descrição* — Adiciona um utilizador à lista de contactos do cliente utilizando para tal o seu identificador.

Apêndice B

Resultados de Invocação do Serviço de Meteorologia

Este apêndice apresenta a diferenciação entre os resultados obtidos do fornecedores *Google Weather* e *Yahoo Weather* de forma a salientar a necessidade de uniformizar os resultados a devolver ao utilizador.

```
<!--...-->
<yweather:location city="Almada" region="" country="PO"/>
<yweather:units temperature="F" distance="mi" pressure="in" speed="mph"/>
<yweather:wind chill="88" direction="120" speed="5" />
<yweather:atmosphere humidity="31" visibility="6.21" pressure="29.97" rising="0" />
<yweather:astronomy sunrise="7:29 am" sunset="7:24 pm"/>
<!--...-->
<item>
    <title>Conditions for Almada, PO at 4:30 pm WEST</title>
    <geo:lat>38.68</geo:lat>
    <geo:long>-9.15</geo:long>
    <link>
        http://us.rd.yahoo.com/dailynews/rss/weather/Almada__PO/*http://weather.yahoo.com/forecast/POXX0001_f.html
    </link>
    <pubDate>Mon, 28 Sep 2009 4:30 pm WEST</pubDate>
    <yweather:condition  text="Partly Cloudy" code="30" temp="88" date="Mon, 28 Sep 2009 4:30 pm WEST" />
    <!--...-->
    <yweather:forecast day="Mon" date="28 Sep 2009" low="66" high="84" text="Partly Cloudy" code="29" />
    <yweather:forecast day="Tue" date="29 Sep 2009" low="63" high="79" text="PM Showers" code="39" />
</item>
<!--...-->
```

Figura B.1: Extracto do resultado de invocação do *Yahoo Weather*

```
<!--...-->
<forecast_information>
    <city data="Paris, IDF"/>
    <postal_code data="paris,france"/>
    <latitude_e6 data=""/>
    <longitude_e6 data=""/>
    <forecast_date data="2009-09-28"/>
    <current_date_time data="2009-09-28 08:17:10 +0000"/>
    <unit_system data="US"/>
</forecast_information>
<current_conditions>
        <condition data="Fog"/>
        <temp_f data="59"/>
        <temp_c data="15"/>
        <humidity data="Humidity: 77%"/>
        <icon data="/ig/images/weather/fog.gif"/>
        <wind_condition data="Wind: N at 5 mph"/>
</current_conditions>
<!--...-->
```

Figura B.2: Extracto do resultado de invocação do *GoogleWeather*

Apêndice C

Resultados de Invocação do Sistema de Mediação – Mapas

Este apêndice ilustra a abstracção gerada, sob a forma de código, pelo sistema de mediação no sentido de devolver um mapa uniformizado ao utilizador.

```
<script type="text/javascript" src="http://api.maps.yahoo.com/ajaxymap?v=3.8&appid=YD-eR1vwa0_JXm49rDdCIPIDA--">
</script>
<script type="text/javascript">

    function startMap(){

        var map = new YMap(document.getElementById("mymap"));

        startControls(map);

        var latitude = 37.4419;
        var longitude = -122.1419;
        FindCoord(map,latitude,longitude);

        var mapaType = YAHOO_MAP_HYB;
        map.setMapType(mapaType);

        var address = "Vila Nova de Famalicão, Portugal";
        Find(map,address);

        callMarkers(map);
        function1(map);

    }

    function startControls(map){
        map.addTypeControl();
        map.addZoomLong();
        map.addPanControl();
    }

    function FindCoord(map,latitude, longitude){
        var startlocation = new YGeoPoint(latitude,longitude);
        map.drawZoomAndCenter(startlocation, 4);
    }

    function Find(map,address){
        map.drawZoomAndCenter(address,4);
    }

    function Marker(map,latitude,longitude,descricao){
        var geoPoint=new YGeoPoint(latitude,longitude);
        var newMarker= new YMarker(geoPoint);
        newMarker.addAutoExpand(descricao);
        map.addOverlay(newMarker);
    }

</script>
```

Figura C.1: Exemplo do bloco de código devolvido como resposta à invocação do método *getMapScript()* por intermédio do Mediador de Mapas

Apêndice D

Estruturas Abstractas Construídas na Concepção dos Mediadores de Serviços

Este apêndice pretende apresentar a composição das estruturas abstractas constituídas na elaboração do mediador de fotografias. Neste sentido, é apresentada apenas uma pequena representação da estrutura real.

```
<list_albuns>
    <album>
        <id></id>
        <title></title>
        <date></date>
        <description></description>
        <photos_left></photos_left>
        <photos_used></photos_used>
        <link></link>
        <media_thumbnail></media_thumbnail>
    </album>
    <!--...-->
</list_albuns>
```

Figura D.1: Estrutura de Abstracção para álbuns utilizada para devolução de resultados no Serviço de Fotografia

```
<list_photos>
    <photo>
        <id></id>
        <album_id></album_id>
        <title></title>
        <date></date>
        <description></description>
        <media_thumbnail></media_thumbnail>
        <camera_model></camera_model>
        <link></link>
        <geo_location>
            <latitude></latitude>
            <longitude></longitude>
        </geo_location>
    <photo>
    <!--...-->
</list_photos>
```

Figura D.2: Estrutura de Abstracção para fotos utilizada para devolução de resultados no Serviço de Fotografia