FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

HUMS vs MMS SOAP Interface and Simulator

João Pedro Magalhães do Vale

Relatório de Projecto

Mestrado Integrado em Engenharia Informática e Computação

Orientador: João Correia Lopes (Professor Auxiliar)

Julho de 2008

HUMS vs MMS SOAP Interface and Simulator

João Pedro Magalhães do Vale

Relatório de Projecto
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Pedro Alexandre Ferreira de Souto (Professor Auxiliar da FEUP)

Arguente: Artur Pereira (Professor Auxiliar da Universidade de Aveiro)
Vogal: João Correia Lopes (Professor Auxiliar da FEUP)

18 de Julho de 2008

Resumo

A Critical Software foi encarregada pela AgustaWestland de actualizar uma aplicação de monitorização do uso e estado da frota de helicópteros EH-101 Merlin da Força Aérea Dinamarquesa, o DGS, para funcionar com um novo sistema de gestão de manutenção. Este projecto consiste em desenvolver a biblioteca responsável pela comunicação com o novo sistema através de *Web Services*, o *SOAP Adaptor*, possibilitando também a ligação a uma base de dados local. Isto deve ser feito sem alterar o funcionamento do DGS. Está também incluída no projecto uma aplicação, o *SOAP Client*, para testar a biblioteca e que permite também criar a base de dados local já referida, povoando-a com informação retirada do sistema de gestão de manutenção.

O projecto foi desenvolvido sobre a plataforma .NET, em C#, recorrendo ao Microsoft Visual C# 2008 que, devido ao seu *interface designer*, facilitou particularmente o desenvolvimento da interface gráfica do *SOAP Client*. O resultado final do projecto foi testado usando os testes de aceitação da versão anterior do DGS, podendo-se assim verificar que o funcionamento da aplicação foi mantido. O *SOAP Client*, sendo apenas uma ferramenta de apoio e não um *deliverable* formal do projecto, foi apenas testado informalmente ao longo do desenvolvimento.

O projecto decorreu a bom ritmo apesar do desenvolvimento se ter atrasado um pouco sem, contudo, ter afectado o prazo final. Chegou-se à conclusão que uma Arquitectura Orientada a Serviços é uma boa forma de atingir um bom nível de integração e que os *Web Services* são de facto uma boa tecnologia para se implementar este tipo de arquitecturas. Isto deve-se principalmente ao facto dos *Web Services* não introduzirem novas tecnologias, limitando-se a reutilizar tecnologias maduras já existentes para atingir um novo fim. Apesar disso, os *Web Services* não são a solução final para problemas de integração, como foi demonstrado pelo facto de ter sido necessário adaptar o código desenvolvido na plataforma .NET para conseguir interagir com um serviço disponibilizado por um servidor aplicacional específico (neste caso, o BEA WebLogic). O *SOAP Client* revelou-se extremamente útil para testar e detectar falhas, tanto no código desenvolvido como no próprio *Web Service*, que seriam complicadas de identificar apenas através do DGS.

Como perspectivas de trabalho futuro, temos a experiência de usar REST em vez de SOAP como o protocolo de troca de mensagens do *Web Service* e redesenhar a base de dados local. O uso de REST parece promissor no sentido em que poderá trazer um melhor desempenho na comunicação, aliado a uma maior simplificação do código. A reestruturação da base de dados, ao eliminar redundâncias de dados, simplificaria algumas das operações, trazendo provavelmente uma melhoria de desempenho.

Abstract

Critical Software was hired by AgustaWestland to update DGS, an application used by the Royal Danish Air Force to monitor the health and usage of its EH-101 Merlin helicopter fleet. The purpose of this update is to make the aplication work with their new maintenance management system. This project consists of developing the library responsible for the communication with the new system using Web Services, called SOAP Adaptor, also supporting the use of a local database, without changing the way DGS works. Also included in the project is an application, *SOAP Client*, whose purpose is to test the communication library and to create the local database, populating it with data retrieved from the maintenance management system.

The project was developed on top of the .NET platform, in C#, using Microsoft Visual C# 2008. Due to its interface designer, this greatly simplified the development of *SOAP Client*'s graphical interface. The project's outputs were tested using the acceptance tests specified for the previous version of DGS, ensuring its functionality remains unchanged. Since the *SOAP Client* was more of a support tool than a formal project deliverable, it was tested informally during the project's development.

The project developed at a good pace in spite of a minor delay entering the test phase. This, however, did not affect the final deadline. It was concluded that a Service-Oriented Architecture is a good way to achieve systems integration and that Web Services are a good technology to implement this type of architecture. This is mainly because Web Services do not bring any new technologies into the arena, using instead existing and proven technologies to achieve a new goal. In spite of this, Web Services are not the definitive solution for integration problems. This was demonstrated by the fact that some changes were necessary for the developed library to be able to interact with the client's Web Service, which was supported by BEA WebLogic, a J2EE application server. The *SOAP Client* application turned out to be extremely useful to test for flaws in the developed code as well as in the Web Service itself. Using only DGS, these flaws would have been very difficult to track down.

In terms of future work, there are two possible approaches. One is to experiment with the use of REST instead of SOAP as the Web Service's message exchange protocol. REST looks promissing especially because of its performance gains and the simplicity of the code required. The other possibility is redesigning the local database's schema. By eliminating some data redundancy, a number of operations would be simplified, with a likely improvement in performance.

Agradecimentos

Começo por agradecer à Critical Software pelo acolhimento e pelo excelente ambiente de trabalho providenciado para o desenvolvimento do projecto. Quero agradecer também ao António Seiça, orientador da empresa, tutor e *project manager*, por todo o apoio dado ao longo do projecto, constante disponibilidade para tirar dúvidas e abundante *feedback*. Agradeço também ao César Lourenço pelo esclarecimento das dúvidas incessantes na ambientação inicial à empresa e ao projecto, e por todas as sugestões e discussões, fossem elas relacionadas com o projecto ou com os mais variados tópicos. Falta referir ainda o resto da equipa do projecto do DGS: João Pereira, José Rui Simões e Rui Lopes, que também deram o seu contributo para levar este projecto a bom porto e deram um muito bom ambiente às reuniões de projecto.

Fica aqui também um grande agradecimento para o professor João Correia Lopes por ter aceite ser o meu orientador para este projecto. As suas sugestões e críticas construtivas foram sempre bem-vindas e seguidas com todo o gosto e as suas questões incisivas e detalhadas foram de grande utilidade para melhor organizar as ideias. A sua boa-disposição foi também uma agradável constante ao longo de todo o projecto.

Não pode faltar um agradecimento para a minha família, que sempre apoiou a minha opção de me deslocar para Coimbra para realizar este projecto, tendo-me dado todo o apoio que poderia ter desejado.

Finalmente, obrigado a todos os meus amigos e colegas da Faculdade, pelas dúvidas tiradas, pelas sugestões e revisões informais a este documento e, principalmente, pelos momentos de diversão para desanuviar e afastar a mente do trabalho.

Mais uma vez, a todos, muito obrigado.

João Vale

“”Forty-two,” said Deep Thought, with infinite majesty and calm.”

Douglas Adams

Conteúdo

Lista de Figuras

Listagens

Abreviaturas e Símbolos

API	*Application Programming Interface*
DGS	*Danish Ground Station*
DLL	*Dynamic Link library*
GUID	*Globally Unique Identifier*
HUMS	*Health and Usage Monitoring System*
ICD	*Interface Control Description*
MMS	*Maintenance Management System*
RDAF	*Royal Danish Air Force*
SGBD	Sistema de Gestão de Bases de Dados
SQL	*Structured Query Language*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*

Capítulo 1

Introdução

Cada vez mais o mundo actual depende da eficiência com que são tomadas decisões, e esta, por sua vez, está dependente de ter o máximo de informação disponível para se estar ao corrente da situação actual. Os Sistemas de Informação são desenhados para armazenar e disponibilizar informação que seja relevante para o seu objectivo.

Um exemplo de um sistema deste género é um sistema de gestão de manutenção, que tem o propósito de guardar informações relativas ao uso e estado de peças e/ou sistemas, permitindo prever quando será necessário fazer a sua manutenção. Desta forma é possível programar melhor as encomendas de novos componentes, optimizar o uso dos recursos humanos e maximizar a disponibilidade dos sistemas em causa.

Aliado à vasta gama de sensores que actualmente se podem instalar junto dos vários componentes, um sistema deste género poderá conter informações muito detalhadas, que irão ser determinantes na tomada da decisão. A importância que é dada à informação vem também dar grande relevância à integração entre aplicações e sistemas de informação.

1.1 Enquadramento

Este projecto insere-se num projecto de manutenção a cargo da Critical Software SA. O projecto de manutenção envolve adaptar uma aplicação de monitorização do uso e do estado de saúde de uma frota de helicópteros de busca e salvamento da Força Aérea Dinamarquesa, *Danish Ground Station* (DGS, desenvolvida por outra empresa), composta por helicópteros EH-101 Merlin, produzidos pela AgustaWestland[1], sendo esta o cliente da Critical Software. A adaptação consiste em alterar a forma como a aplicação comunica com o sistema de gestão de manutenção, onde são mantidos os registos. O objectivo é que a comunicação se passe a fazer através de *Web Services*.

[1] http://www.agustawestland.com/

A Critical Software é uma empresa de software que fornece soluções, serviços e tecnologias para sistemas *mission-critical* e *business-critical* em várias áreas de mercado. Foi criada em 1998 por três estudantes de doutoramento da Universidade de Coimbra, tendo sido incubada no Instituto Pedro Nunes. A empresa evoluiu rapidamente tendo fechado em 1999 o seu primeiro grande contrato internacional, com a NASA.

A empresa apostou desde cedo na qualidade, tendo obtido diversas certificações como CMMI[2] nível 3 (a primeira empresa em Portugal, estando actualmente a trabalhar na certificação de nível 5), SPICE[3], ISO 9001 TickIt[4] (primeira empresa Ibérica) e AQAP[5] 2110 e AQAP 150 da NATO. Estas, para além de reflectirem o nível dos seus processos internos, também lhe permitem aceder a novos mercados onde estas certificações são um requisito.

Actualmente a empresa continua em franco crescimento, estando em 2007 pela quarta vez consecutiva no *ranking* das 500 empresas europeias com maior crescimento e empregando mais de 400 pessoas nos seus escritórios em Coimbra, Porto, Lisboa, Southampton (Inglaterra), San José (Califórnia, EUA) e, mais recentemente, em Bucareste (Roménia). As suas receitas no ano de 2007 tiveram um aumento notável de 60%, sendo que 70% do seu volume de negócios vem do estrangeiro, espelhando a sua forte presença internacional. Entre os seus clientes incluem-se a NASA, European Space Agency, Siemens, AgustaWestland, Vodafone, European Aeronautic Defence and Space, Honeywell, Soporcel e o grupo PSA.

1.2 Projecto

Este projecto consiste, mais especificamente, no desenvolvimento da biblioteca que irá abstrair a comunicação entre a aplicação e o sistema de gestão de manutenção, sendo as alterações feitas à aplicação em si efectuadas por outro membro da equipa da Critical Software. Esta biblioteca irá funcionar como um *wrapper*, mantendo a API usada anteriormente, mas comunicando com o sistema de gestão de manutenção através de *Web Services*.

Está também no âmbito do projecto o desenvolvimento de uma aplicação para testar o *Web Service*. Esta tem como funcionalidades a possibilidade de chamar os vários métodos do *Web Service* e apresentar os resultados, ler os *logs* do DGS criados durante o uso da aplicação e executar as chamadas na mesma sequência com os mesmos argumentos e criar uma base de dados local a partir da informação disponibilizada pelo *Web Service* para tanto a aplicação como o DGS poderem trabalhar em modo *offline*, ou seja, sem recurso ao *Web Service*.

[2] http://www.sei.cmu.edu/cmmi
[3] http://www.isospice.com/
[4] http://www.tickit.org
[5] http://www.nato.int/docu/standard.htm#AQAP

1.3 Motivação e Objectivos

A motivação por trás deste projecto é a decisão da RDAF de mudar de sistema de gestão de manutenção, forçando esta alteração a modificar as aplicações que dependem dele. No caso do DGS, a modificação para o tornar compatível com o novo sistema foi planeada de forma a que, caso surja uma nova alteração no sistema de gestão de manutenção, o impacto seja mínimo.

A nível mais pessoal, existem três motivações principais para este projecto. A primeira é as tecnologias envolvidas, nomeadamente *Web Services*. O acesso à Internet, ou a pelo menos uma intranet, é cada vez mais ubíquo, sendo portanto o canal ideal para a troca de informação. Os *Web Services*, por sua vez, devido à simplicidade do seu conceito e ao uso das tecnologias em que se baseia a *Web*, são a forma ideal para trocar essa informação. A segunda é a área do projecto em si. Sendo um entusiasta de tecnologia militar, aeronaves em particular, esta foi uma boa oportunidade de lidar, se bem que não muito directamente, com a área. A terceira é a empresa em si. Sendo uma empresa conhecida por ter processos bastante apurados, seria com certeza uma experiência enriquecedora.

Dado que este projecto não é puramente académico, um dos grandes objectivos é chegar ao fim com uma solução robusta que cumpra os requisitos do cliente. Não só porque é essa a função da empresa como prestadora de um serviço, mas também porque é uma oportunidade de experimentar todo o processo de desenvolvimento num contexto mais real, num ambiente menos controlado.

1.4 Estrutura do Relatório

Para além da introdução, este relatório contém mais quatro capítulos. No Capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados. No Capítulo 3 é descrito o problema e é dada uma visão geral da solução. São também apresentados os requisitos para a solução. No Capítulo 4 são descritos em mais detalhes os pormenores mais relevantes da solução. No Capítulo 5 é apresentado um resumo do trabalho efectuado, é apreciada a satisfação dos objectivos e, finalmente, são apresentadas algumas sugestões para possíveis trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Este capítulo tem por objectivo descrever o estado da arte e os trabalhos existentes relacionados com o projecto. São também introduzidas e descritas as diversas tecnologias e princípios arquitecturais passíveis de ser utilizadas no desenvolvimento do projecto, sendo identificados os seus principais pontos fortes e pontos fracos.

Na conclusão são apresentadas e justificadas as decisões tomadas a nível tecnológico, tendo estas tido como base os requisitos do cliente e as características de cada uma das tecnologias.

2.1 Arquitecturas Orientadas a Serviços

Uma Arquitectura Orientada a Serviços (*Service-Oriented Architecture*, SOA) é um tipo de arquitectura que, em termos conceptuais, evoluiu a partir de conceitos como a reutilização, o agrupamento de funcionalidades e o modelo cliente/servidor. A grande diferença de SOA para os métodos mais tradicionais, tais como separar/estruturar funcionalidades por funções/métodos/procedimentos ou por classes, é que esta é feita a mais alto-nível, de forma consideravelmente menos granular. As funcionalidades são agrupadas em **serviços**, em que cada serviço reflecte uma acção real, que pode ser executada múltiplas vezes. Um serviço é algo que uma pessoa, de forma abstracta, pode encarar como um serviço e que por si só traz algum valor acrescentado [Gro06]. SOA é mais orientado à mensagem do que à operação. Exemplos disto são serviços que devolvam o saldo de uma conta bancária, uma previsão meteorológica para uma determinada localização, etc.

Um serviço, a unidade básica do SOA, é uma funcionalidade que é disponibilizada através de uma interface bem conhecida, de forma independentemente do estado de qualquer outro componente do sistema. Este funciona como uma caixa negra, ou seja, quem

acede a esse serviço não sabe (nem precisa de saber) como é que ele funciona internamente. Os serviços podem estar distribuídos tanto logicamente (diferentes entidades a correr na mesma máquina) como fisicamente, comunicando através de uma rede.

Em termos tecnológicos, SOA é *loosely-coupled*, ou seja, cada entidade assume muito pouco em relação à outra, principalmente em relação à tecnologia usada na implementação, não estando dependente de nenhuma em particular, tanto do lado do cliente como do lado do servidor, ou até entre pares. Por exemplo, normalmente um servidor e cliente RPC são implementados em C, no caso de RMI em Java, etc. Esta é a razão pela qual *Web Services* são bastante usados para implementar arquitecturas SOA, visto que o único requisito é que partilhem o mesmo canal de transporte (usualmente HTTP). Em SOA a interoperabilidade entre as entidades é baseada numa descrição formal do serviço e partindo daí pode-se ter o servidor implementado na linguagem X, um cliente a usar a plataforma Y e ainda outro a usar a linguagem Z.

Contudo, qualquer uma das tecnologias referidas (e mais, como por exemplo CORBA e DCOM) pode ser usada para implementar uma arquitectura SOA [Wik08b], desde que usada de acordo com os princípios desta: sendo mais orientada às mensagens e sendo as funcionalidades baseadas em serviços e não em operações. Por exemplo, uma forma de estender o tempo de vida útil de aplicações-legado, seja qual for a tecnologia usada, é construindo um *wrapper* à sua volta que adapte o seu funcionamento para o modelo de serviços.

2.2 *Web Services*

Actualmente, uma das formas mais comuns de implementar uma arquitectura SOA é através de *Web Services*. O *World Wide Web Consortium*[1] (W3C) define *Web Services* como um sistema de software concebido para interacções entre máquinas, de forma interoperável, através de uma rede informática [Gro04].

Um *Web Service*, como o nome deixa transparecer, é um serviço que é disponibilizado através da *Web*, ou seja, pela Internet (ou por uma intranet). Isto traz uma grande vantagem relativamente a tecnologias como CORBA, DCOM ou RMI porque pode usar diversos protocolos de transporte como, por exemplo, o HTTP, FTP, Jabber, SMTP ou POP3. Contudo, o protocolo mais usado é o HTTP dado que este é um protocolo simples e maduro, tendo a versão 1.0 surgido em 1996 e tendo sido actualizado ao longo dos anos. É também um protocolo que está muito disseminado, estando disponível em virtualmente todas as plataformas usadas actualmente. Do ponto de vista de redes locais / redes empresariais, as *firewalls* já costumam estar configuradas de forma a deixar sair o tráfego HTTP e, caso já exista um servidor *Web* na rede interna (algo bastante comum), não é necessário qualquer configuração extra. Isto, claro, aplica-se no caso de serviços disponibilizados

[1] http://www.w3.org

para o exterior da rede. No caso de serviços disponíveis apenas internamente, a questão nem se levanta. Do ponto de vista da segurança, o protocolo contempla autenticação e permite o uso de ligações seguras através de *Secure Socket Layer* (SSL) ou de *Transport Layer Security* (TLS). Tem ainda a vantagem de suportar, de forma transparente, a compressão dos dados transmitidos, o que reduz consideravelmente a largura de banda necessária para operar o serviço [FGM+99]. Esta é uma das grandes vantagens dos *Web Services*: em vez de trazer novas tecnologias, adapta as tecnologias existentes para criar algo de novo.

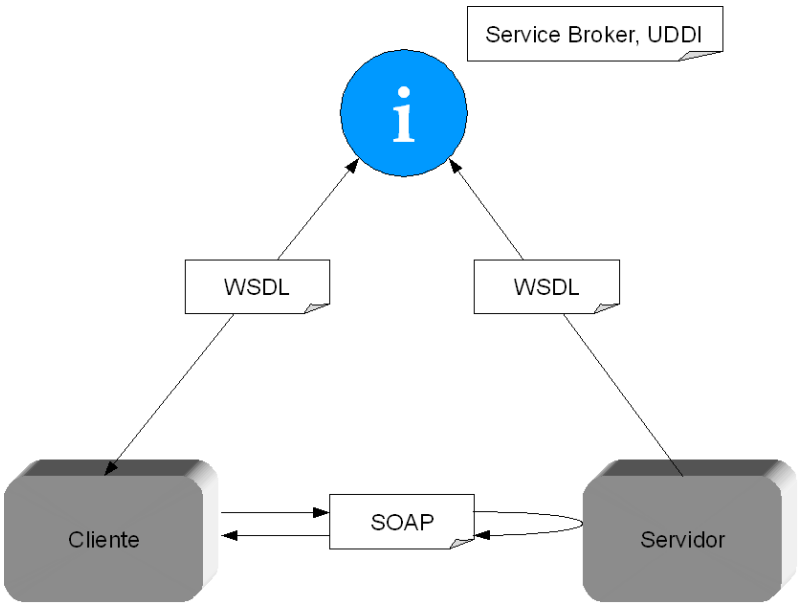

Figura 2.1: Arquitectura de um *Web Service*

O modo de funcionamento de um *Web Service* é semelhante ao de outras tecnologias de distribuição [Wik08c]. O cliente usa uma descrição do serviço disponibilizado para criar uma representação local (*proxy*), usando depois esse *proxy* para invocar os métodos oferecidos pelo serviço. No caso dos *Web Services*, essa descrição é feita recorrendo à *Web Services Description Language* (WSDL), sendo normalmente este também o nome dado ao documento que contém a descrição. A WSDL é uma linguagem, baseada em XML, que indica onde se encontra um determinado *Web Service* (o URL onde este se encontra) e como o utilizar. Isto inclui os métodos disponibilizados, bem como os parâmetros que recebe e os valores que devolve. A invocação é feita com um pedido HTTP POST, ou seja, o pedido é enviado no corpo da pedido. Do lado do servidor, o pedido POST é recebido e inicialmente tratado como se fosse um pedido normal de uma página *Web*. Depois de tratada a mensagem, é feita a chamada com os respectivos parâmetros e a resposta é devolvida no corpo da resposta HTTP.

Como as outras tecnologias de distribuição, os *Web Services* também contemplam

uma forma de pesquisar serviços (*discovery*): o *Universal Description, Discovery and Integration* (UDDI). Esta especificação, também baseada em XML, define um sistema de registo de serviços (análogo ao *RMI Registry* em Java), em que um servidor se regista num serviço de registo (*Service Broker*) com a informação relevante para a sua identificação, assim como o documento XML com a sua WSDL. Posteriormente, um cliente acede ao registo UDDI, procura o serviço desejado e descarrega a WSDL para o poder utilizar. Quando a especificação do UDDI foi criada, a visão dos seus autores era que o seu funcionamento fosse semelhante ao das Páginas Amarelas, em que as empresas registavam os seus serviços num registo globalmente acessível. Entre 2000 e 2006 existiu uma iniciativa desse género, o *Universal Business Registry*, que era composto por um registo principal e um conjunto de nós, cujos operadores incluíam a IBM, SAP e Microsoft. Estes nós foram progressivamente desactivados pelos operadores, que entendiam que já não eram necessários visto terem servido mais como implementações de referência para impulsionar a especificação do que para fazer negócio[Des05].

2.2.1 SOAP

SOAP é um protocolo para a troca de mensagens em formato XML através de uma rede de computadores. Usualmente o termo é incorrectamente usado como um sinónimo de *Web Service*, sendo o SOAP apenas uma das formas de implementar a troca de mensagens de um *Web Service* (outra forma é o REST, que será apresentado na secção seguinte). Isto deve-se ao SOAP ser usado como a referência para o protocolo de troca de mensagens em *Web Services*, aparecendo até na própria definição [Gro04] que as mensagens trocadas são mensagens SOAP. Contudo, o próprio documento de especificação [W3C07] refere (ponto 2.3.1.3.3) que as mensagens podem ser SOAP, mas também podem ser XML simples ou até os próprio parâmetros do URL usado no pedido HTTP.

Uma mensagem SOAP é composta por um *envelope*, que contém dois elementos: o cabeçalho e o corpo da mensagem. O cabeçalho é principalmente usado para, quando desejado, estender a especificação do SOAP. Estas extensões, aprovadas pela W3C ou por outros organismos, acrescentam funcionalidades como anexos às mensagens, fiabilidade na entrega das mensagens, transacções, etc. O corpo da mensagem contém o método chamado mais os respectivos parâmetros ou a resposta, conforme seja o pedido ou resposta do servidor. Apresenta-se seguidamente um pedido e uma resposta SOAP como exemplo[2]:

```
1 POST /InStock HTTP/1.1
2 Host: www.example.org
3 Content-Type: application/soap+xml; charset=utf-8
4 Content-Length: nnn
```

[2]Exemplo retirado de http://www.w3schools.com/SOAP/soap_example.asp

```
5
6  <?xml version="1.0"?>
7  <soap:Envelope
8  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
9  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
10
11   <soap:Body xmlns:m="http://www.example.org/stock">
12     <m:GetStockPrice>
13       <m:StockName>IBM</m:StockName>
14     </m:GetStockPrice>
15   </soap:Body>
16
17 </soap:Envelope>
```

Listagem 2.1: Pedido SOAP

Esta mensagem faz um pedido a um *Web Service* fictício (http://www.example.org/stock). O pedido consiste numa chamada ao método *GetStockPrice*, cujo parâmetro *GetStockPrice* tem o valor *IBM*. Estruturalmente, está organizada da seguinte forma:

- Linhas 1 a 4: cabeçalhos HTTP;
- Linhas 6 a 17: mensagem SOAP (*envelope*);
- Linhas 8 e 9: cabeçalhos SOAP;
- Linhas 11 a 15: corpo da mensagem SOAP;
- Linhas 12 a 14: chamada efectuada;
- Linha 13: parâmetros da chamada.

```
1  HTTP/1.1 200 OK
2  Content-Type: application/soap+xml; charset=utf-8
3  Content-Length: nnn
4
5  <?xml version="1.0"?>
6  <soap:Envelope
7  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
8  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
9
10   <soap:Body xmlns:m="http://www.example.org/stock">
11     <m:GetStockPriceResponse>
12       <m:Price>34.5</m:Price>
13     </m:GetStockPriceResponse>
14   </soap:Body>
15
16 </soap:Envelope>
```

Listagem 2.2: Resposta SOAP

Esta mensagem é a resposta ao pedido enviado na Listagem 2.1. A resposta vem dentro de *GetStockPriceResponse*, sendo composta apenas por um elemento, sendo o seu valor *34.5*.Estruturalmente, está organizada da seguinte forma:

- Linhas 1 a 3: cabeçalhos HTTP;
- Linhas 6 a 16: mensagem SOAP (*envelope*);
- Linhas 7 e 8: cabeçalhos SOAP;
- Linhas 10 a 14: corpo da mensagem SOAP;
- Linhas 11 a 13: chamada efectuada;
- Linha 12: parâmetros da chamada.

Este tipo de sintaxe das mensagens torna-as decifráveis por humanos e torna-as também relativamente simples de tratar programaticamente. Esta sintaxe é também, contudo, a origem da maioria das desvantagens do SOAP. O processamento do XML, especialmente considerando um volume de tráfego mais elevado ou uma rápida sequência de pedidos, introduz um *overhead* considerável em termos de processamento. O uso de pedidos HTTP POST também impede que chamadas frequentes sejam guardadas em *cache* de servidores *proxy* ou outros meios de *caching*.

2.2.2 REST

O REST (*REpresentational State Transfer*) é uma arquitectura para troca de hipermédia[3] entre sistemas distribuídos introduzida por Roy Fielding na sua tese de doutoramento em 2000 [Fie00]. Esta arquitectura consiste num conjunto de princípios para definir recursos e aceder-lhes através de uma rede [Wik08a]:

- cada recurso tem um identificador único;
- existe um conjunto restrito de operações e de tipos de conteúdos;
- existe uma interface uniforme para interagir com os recursos;
- permite uma estrutura por camadas (graças à interface uniforme), em que pode existir um número arbitrário de camadas, mas cada uma só conhece aquela com quem interage;
- o acesso é feito segundo o modelo cliente-servidor;
- não é mantida informação sobre o estado da comunicação (é *stateless*).

[3] http://en.wikipedia.org/wiki/Hypermedia

Um exemplo de uma arquitectura REST em funcionamento é a própria *World Wide Web*, o que não é surpreendente dado que Roy Fielding também esteve envolvido na definição da especificação de *standards* HTTP, URI e HTML. A *Web* é baseada em recursos (URLs) com hiperligações entre eles, cada recurso é único, existe um conjunto restrito de operações (os métodos HTTP), existe diferenciação entre tipos de conteúdo (tipos MIME[4]), funciona num modelo cliente-servidor, não mantém estado, funciona por camadas (podem existir *gateways*, *proxys*, *firewalls*, etc entre o cliente e o servidor) e possibilita o recurso a *caches* devido ao grande uso dado ao método HTTP GET.

O REST como um todo não pode ser comparado com SOAP visto ser uma arquitectura e não um protocolo de troca de mensagens. Contudo, os seus princípios podem ser aplicados ao conceito de *Web Service* se interpretados num sentido mais lato, referindo-se apenas a uma forma simples de transmitir dados específicos de um determinado contexto através de HTTP. As próximas referências a REST que surgirem irão referir-se a esta definição e não à definição mais formal.

Uma descrição mais *ad-hoc* de REST aplicado a *Web Services* é que este se baseia fortemente em URLs (num contexto não-*Web* seria um URI, *Uniform Resource Identifier*), tendo as diferentes chamadas URLs únicos. No caso de SOAP, as chamadas têm todas o mesmo URL, sendo usada a mensagem SOAP para indicar qual a chamada pretendida. Um exemplo de uma chamada REST para saber o preço de um livro numa loja *online* através do seu número ISBN pode ser:

```
http://example.org/bookPrice?ISBN=123456789
```

ou

```
http://example.org/bookPrice/123456789
```

O uso do método HTTP GET, que é usado apenas para leitura de informação, faz com que seja possível fazer *cache* dos resultados, algo que o método HTTP POST usado por SOAP não permite devido à própria natureza das mensagens, em que toda a informação do pedido vai no corpo da mensagem. Comparativamente com SOAP, REST tem a eventual desvantagem (depende do contexto da aplicação) de, ao não ter forçosamente uma estrutura bem definida para as respostas nem um documento comparável a uma WSDL, não lhe conferir o *strong-typing* necessário no contexto de aplicações distribuídas. Contudo, a especificação WSDL 2.0 já contempla métodos HTTP para além de GET e POST (como o PUT e o DELETE), o que facilita a descrição de serviços REST. Quanto à ausência de estrutura das mensagens, isto depende da implementação, visto que estas podem ter uma estrutura qualquer. Contudo, a ausência de uma estrutura relativamente pesada de cabeçalhos e um determinado formato para o corpo da mensagem faz com

[4] http://en.wikipedia.org/wiki/MIME

que se poupe tanto em processamento (processar constantemente uma grande quantidade de texto torna-se dispendioso), como em largura de banda visto que, evitando o formato SOAP, uma resposta pode chegar a ser dez vezes mais pequena[Asa08]. O facto de os pedidos usarem principalmente o método HTTP GET também permite o uso de *caches*, tanto a nível do cliente como a nível do servidor, ou até intermediários, como por exemplo um servidor *proxy*. Outra vantagem é que, por ser bastante simples, também simplifica a implementação de clientes e serviços que usem REST.

Contudo, REST não é algo que possa substituir SOAP. Por exemplo, em situações em que a segurança seja importante, o REST não é tão desejável visto passar os parâmetros através do URI, ficando assim expostos dados potencialmente sensíveis. Esta característica de REST tem ainda a desvantagem de limitar o tamanho dos pedidos. Por exemplo, o servidor HTTP mais usado actualmente, o Apache[5], tem como limite predefinido para o tamanho de um URL 8 KBytes, o que limita seriamente a capacidade de enviar dados binários.

2.3 Ferramentas e Outras Tecnologias

Este projecto envolve um grande número de tecnologias. O DGS está implementado em OpenROAD[6], não sendo portanto surpreendente que o SGBD usado para a base de dados de HUMS seja o Ingres dado que existe uma grande integração entre os dois. O sistema de gestão de manutenção é baseado em Oracle, sendo a comunicação entre este e o DGS mediada por componentes em C++ .

Inicialmente, um outro membro da equipa iniciou a implementação da biblioteca de comunicação por *Web Services* em C++ , tendo optado por mudar para C# devido a esta última ter uma interacção mais fácil com *Web Services*.

Como ferramenta de desenvolvimento foi escolhido o Microsoft Visual C# 2008 Express Edition[7]. Esta escolha prende-se com o facto de o Visual Studio, para além de ser o IDE de referência para a linguagem, ter um bom suporte para *Web Services*, incluindo geração de código a partir de uma WSDL.

2.4 Conclusões

No contexto de integração de sistemas de gestão de manutenção com *ground stations* não existem trabalhos relacionados, pelo menos disponíveis publicamente. É compreensível dado que é uma área muito específica e o mundo das aplicações militares não costuma ser muito aberto. Abstraindo o contexto e simplificando o problema, trata-se

[5] http://httpd.apache.org
[6] http://www.ingres.com/products/openroad.php
[7] http://www.microsoft.com/express/vcsharp/

de converter a camada de acesso a dados de uma aplicação, com o requisito principal de manter as chamadas e as suas assinaturas, não sendo também tema a que tenha sido dedicada muita pesquisa.

Relativamente à decisão de usar SOAP ou REST, o uso de SOAP era um requisito do cliente e, dado os prazos curtos e o facto de o projecto já ter arrancado, não havia tempo para estudar de forma aprofundada uma nova tecnologia, apresentá-la ao cliente e, eventualmente, esta passar por todo o processo de aprovação, mais a própria alteração no projecto. Contudo, a análise feita neste trabalho a ambas as tecnologias, apesar de não ter sido baseada em comparações analíticas (desempenho, carga, etc), permite concluir que REST seria uma boa opção, porque apesar do serviço não estar disponível publicamente, não tendo portanto um grande número de utilizadores concorrentes, certos casos de uso da *ground station* implicam um grande número de pedidos seguidos, pelo que o aumento de desempenho seria bem vindo. Também no caso da aplicação estar a ser usada em teatro de operações, a largura de banda poderá ser mais limitada, sendo também a menor quantidade de informação trocada uma mais valia. A simplicidade (em termos de código) de implementar um serviço REST não traz vantagens neste caso visto a plataforma usada (Microsoft .NET) ter ferramentas que facilitam consideravelmente o uso de SOAP. No que respeita à segurança, optou-se por usar autenticação HTTP em vez de extensões de SOAP, tornando assim esse factor irrelevante para a comparação. A ausência de uma maneira mais formal de descrever o serviço em REST, apesar da existência da especificação WSDL 2.0, não é uma desvantagem visto que a API da aplicação está bem definida na documentação do projecto e os clientes do serviço são todos conhecidos e controlados pela mesma entidade, não sendo assim necessário algo semelhante a uma WSDL para manter um certo *strong-typing*.

Em termos de plataforma de desenvolvimento, C# é uma boa escolha visto que o melhor desempenho oferecido pelo C++ não é relevante para o caso de uso desta biblioteca. São também vantagens do C# a facilidade e simplicidade de interagir com *Web Services* e o facto de ser código *managed*, o que facilita a escrita de código

Capítulo 3

Descrição do Problema

Este capítulo começa por expor o problema tratado no projecto, fazendo uma descrição do funcionamento actual do sistema, das necessidades de alteração e do funcionamento da solução proposta. Em seguida são apresentados os requisitos para os dois componentes do projecto e, finalmente, um resumo juntamente com as conclusões.

3.1 Problema e Arquitectura da Solução

O sistema alvo do projecto é constituído por três módulos:

On Board System Sistema a bordo do helicóptero que monitoriza em tempo real um conjunto de sensores, recolhendo informações sobre as condições de voo;

***Health and Usage Monitoring System* (HUMS)** Também conhecida por *ground station*, é a aplicação de recolha de dados em terra que permite, a partir dos dados recolhidos pelo *On Board System*, calcular a vida útil do helicóptero, de cada sistema e de cada peça em função da sua utilização e de acordo com factores de desgaste;

***Maintenance Management System* (MMS)** Sistema de gestão de manutenção que permite gerir de forma eficiente a manutenção da frota em termos de disponibilidade, logística de peças e recursos humanos.

Os sensores a bordo do helicóptero recolhem dados que são guardados num cartão de memória. Após o voo, os dados recolhidos são importados para a aplicação de HUMS através de um leitor de cartões apropriado. Após o processo de recolha dos dados do cartão é feito o seu tratamento, sendo criados eventos relativos a possíveis situações anómalas e calculado o desgaste de cada componente do helicóptero. A informação sobre o voo é guardada numa base de dados local (*HUMS database*) e os restantes dados são

guardados no MMS. Este processo é chamado um *debrief*. Os detalhes de todos os voos aos quais foram feitos *debriefs* podem também ser consultados através da aplicação. A informação pode ainda ser exportada para ser enviada ao fabricante dos helicópteros. Isto pode ser visualizado na Figura 3.1.

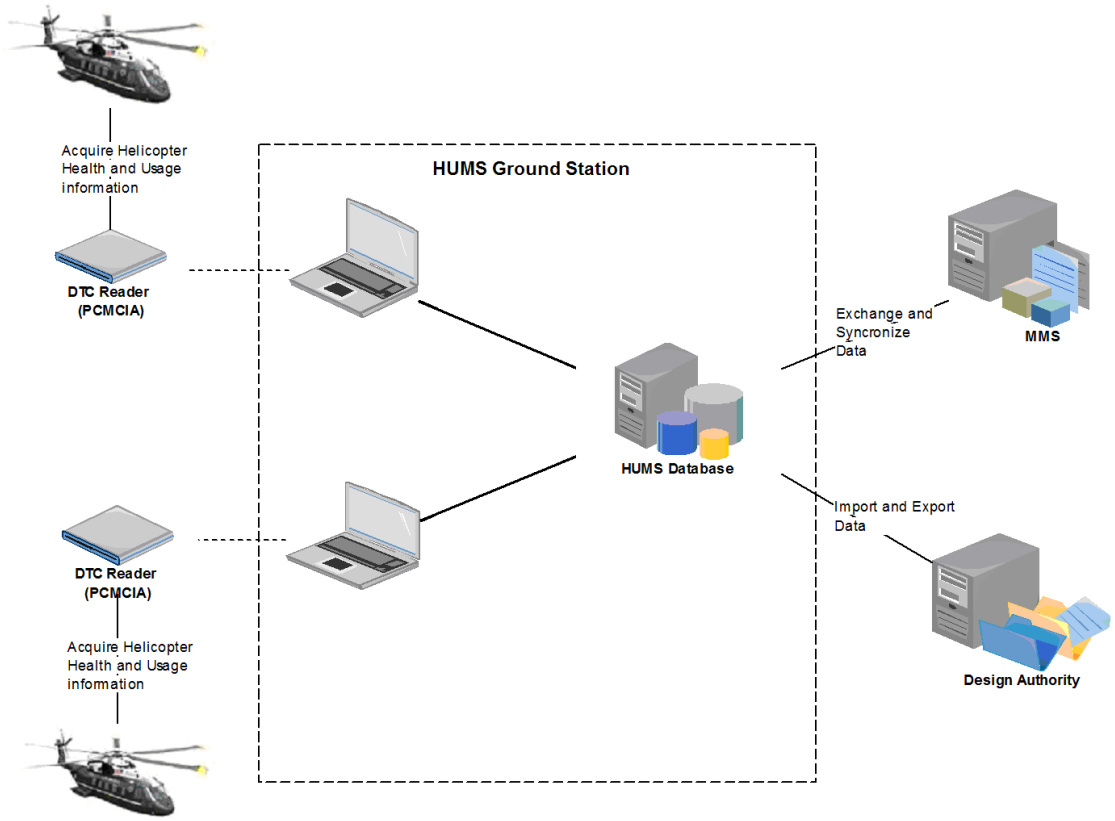

Figura 3.1: Vista do Sistema

A arquitectura da versão actual da aplicação de HUMS, o DGS 1.2, pode ser vista na Figura 3.2. O DGS interage a partir de um dos seus componentes, o *HUMS Abstraction Layer* (HAL), que comunica directamente com a base de dados do MMS através de um componente próprio, o *OO4O*.

O MMS existente é uma solução desenvolvida *in-house* suportada em Oracle[1], que foi decidido abandonar para adoptar uma solução *Commercial Off-The-Shelf* (COTS) da SAP[2], cuja implementação estará a cargo da Siemens alemã[3]. Tendo a aplicação de HUMS sido desenhada para o MMS original, revelou-se necessário re-implementar a componente de comunicação de forma a esta poder interagir com o novo MMS. Contudo, para evitar que esta situação volte a ocorrer, foi decidido tornar esta comunicação

[1] http://www.oracle.com
[2] http://www.sap.com
[3] http://www.siemens.de

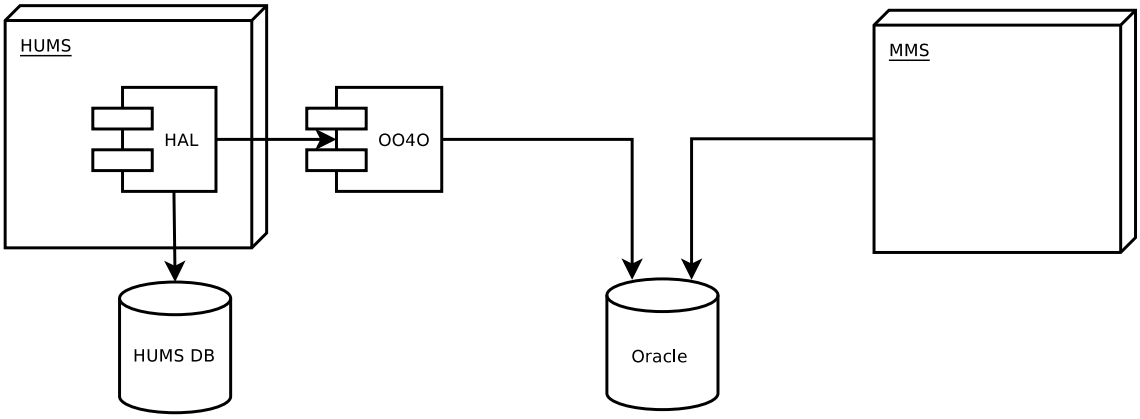

Figura 3.2: Arquitectura do DGS 1.2

independente do software de MMS, tendo-se optado por uma arquitectura baseada em *Web Services*. O MMS disponibiliza um *Web Service* que a aplicação de HUMS usa para submeter dados de novos voos, assim como para aceder aos dados de voos anteriores dos vários helicópteros que estão a ser monitorizados.

A versão que se pretende desenvolver, o DGS 1.3, irá adaptar o HAL para usar uma biblioteca, denominada *SOAP Adaptor*, que abstrai a comunicação com o *Web Service*, mantendo as funções, parâmetros e valores de retorno do *OO4O*. O *SOAP Adaptor* irá comunicar com o *SAP Broker*, um componente do SAP, que usa o servidor aplicacional J2EE WebLogic[4]. As diferenças em termos de arquitectura podem ser vistas na Figura 3.3.

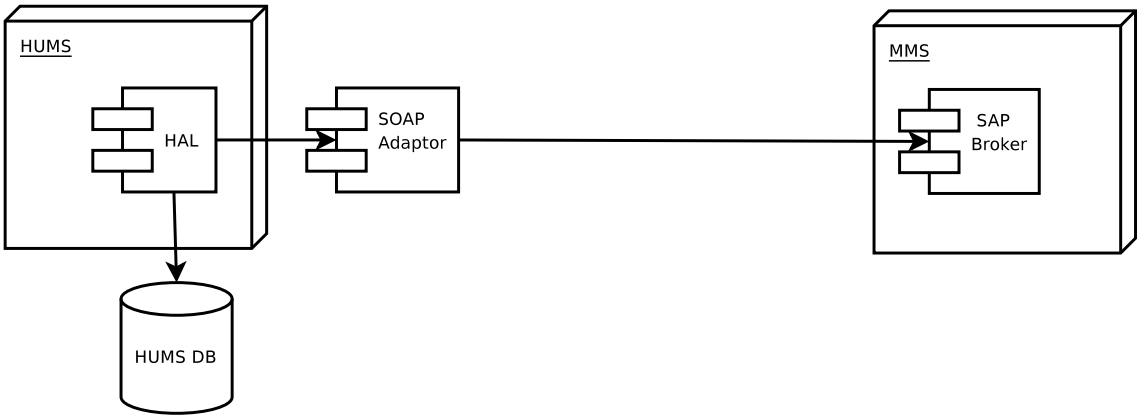

Figura 3.3: Arquitectura do DGS 1.3

Adicionalmente, para efeitos de teste, é também necessária uma aplicação que permita testar individualmente cada chamada, reproduzir uma sessão de utilização do DGS

[4] http://www.bea.com

(através dos *logs* que este cria) e criar uma cópia local da base de dados em uso, a base de dados *stub*. Esta aplicação, o *SOAP Client* usará a mesma biblioteca que o DGS para aceder ao serviço. A sua arquitectura pode ser vista na Figura 3.4.

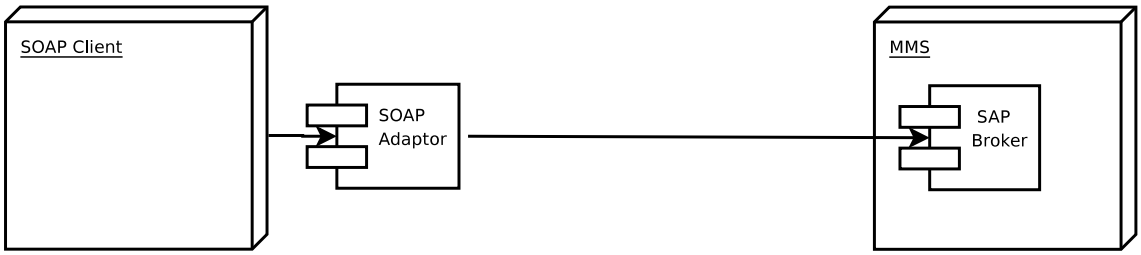

Figura 3.4: Arquitectura do *SOAP Client*

O teste individual a cada chamada é essencial para detectar possíveis erros que apenas através do uso do DGS seriam extremamente complicados de encontrar. A reprodução de uma sessão de uso do DGS é de grande utilidade tanto para testar a solução tendo como base uma sessão real de utilização como para, aliada a uma cópia local da base de dados em uso, investigar qualquer problema mais difícil de identificar. Isto é uma mais-valia visto que se trata de uma aplicação *data-centric*, sendo portanto sensível a variações nos dados. A cópia local permite, para todos os efeitos, reproduzir todo o ambiente no momento em que o problema surgiu. Por esta razão é também necessário que o *SOAP Adaptor* permita que seja feita uma ligação a uma base de dados, mantendo o mesmo comportamento que teria ligado a um *Web Service*. A cópia local pode ainda servir para tornar possível utilizar o DGS ou o *SOAP Client* em cenários onde obter uma ligação ao MMS principal se revele difícil ou indesejável (quando se encontrar em teatro de operações, por exemplo). Contudo, este último não será o principal caso de uso visto que implicaria todo um processo de avaliação e certificação, que nesta fase não é um dos objectivos do projecto. Note-se que a cópia local não é uma cópia directa da base de dados original, é apenas retirada a informação necessária para o funcionamento da aplicação, relativamente aos helicópteros que a *ground station* está a controlar. Do ponto de vista de ambas as aplicações, nada muda quando se usa uma base de dados em vez de um *Web Service*, como se pode ver na Figura 3.5 e na Figura 3.6.

3.2 Requisitos

Em seguida são apresentados os requisitos para o *SOAP Adaptor* e o *SOAP Client*. Dado que o *SOAP Client* usa o *SOAP Adaptor* para a comunicação, vai ser apresentado primeiro o *SOAP Adaptor*.

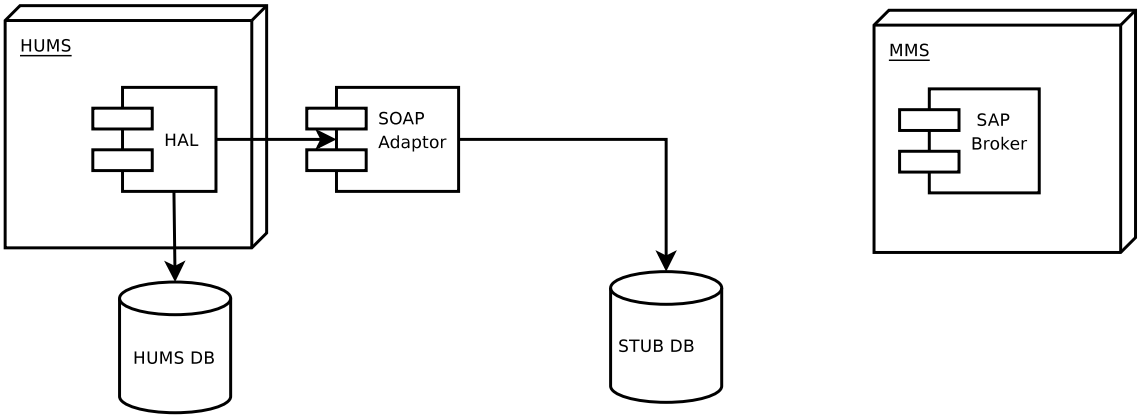

Figura 3.5: Arquitectura do DGS 1.3 a Usar uma Base de Dados *Stub*

3.2.1 *SOAP Adaptor*

Implementar a API disponibilizada pelo *OO4O* O módulo deve disponibilizar as mesmas funções oferecidas pelo *OO4O*, com os mesmos parâmetros. Os valores de retorno poderão ser encapsulados desde que a informação neles contida seja a mesma. Deverão ser tidas em conta as alterações que forem propostas pelo ICD, documento a ser produzido pela Critical Software, AgustaWestland, RDAF e Siemens.

Permitir interacção com um *Web Service* e com uma base de dados A biblioteca deverá permitir, de forma transparente para a aplicação que a utiliza, ligar-se e utilizar os dados provenientes tanto do *Web Service* como de uma base de dados local (através de ODBC[5]).

Suportar ligações a bases de dados Oracle e Ingres Inicialmente vai ser usado Oracle visto ser este o SGBD usado originalmente. Contudo, para a solução final pretende-se usar Ingres visto ser este o SGBD usado para a base de dados de HUMS, evitando um segundo SGBD a consumir recursos da máquina.

Exportar tipos de dados usados Exportar tipos de dados necessários para outras aplicações (o HAL em particular) poderem interpretar correctamente o resultado das chamadas

Permitir manter um registo, em texto, de toda a actividade Este ficheiro de registo, em texto simples, segue o mesmo formato que os *logs* do DGS.

[5] http://en.wikipedia.org/wiki/ODBC

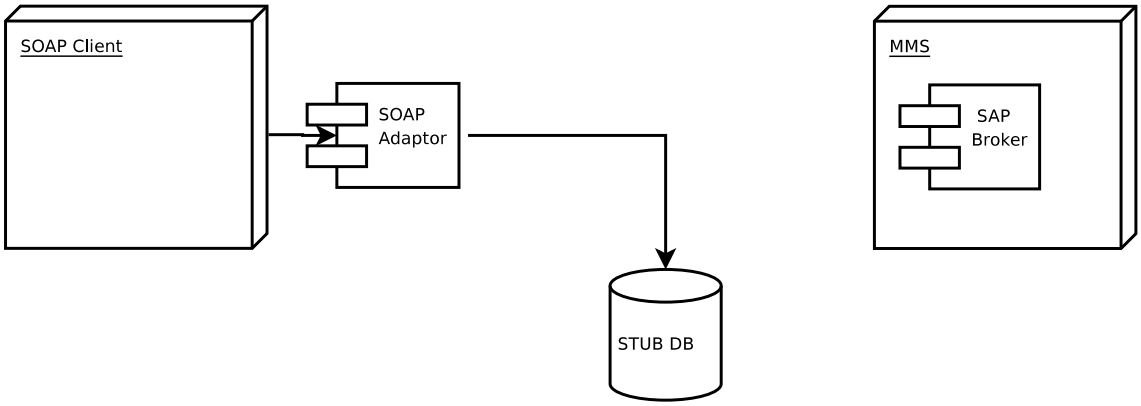

Figura 3.6: Arquitectura do *SOAP Client* a Usar uma Base de Dados *Stub*

3.2.2 *SOAP Client*

Ligar ao *Web Service* Dado o URL de um *Web Service* e uma combinação válida de utilizador e palavra-passe, estabelecer uma ligação com esse *Web Service*, com suporte opcional para um servidor *proxy* e para escolher entre HTTP versão 1.0 e versão 1.1.

Ligar a uma base de dados Dado uma ligação ODBC válida e uma combinação válida de utilizador e palavra-passe para essa mesma ligação, estabelecer uma ligação com uma base de dados.

Efectuar chamadas aos métodos Será possível efectuar as chamadas ao serviço especificado no momento do *login* inicial, especificando os parâmetros e sendo o resultado da chamada visualizado numa área apropriada.

Visualizar resultados Estará sempre visível na aplicação uma área de visualização que irá registar as chamadas com os respectivos parâmetros, as respostas às chamadas e quaisquer informações auxiliares que sejam relevantes.

Gravar área de visualização Será possível guardar os conteúdos da área de visualização num ficheiro de testo simples.

Limpar área de visualização Será possível limpar os conteúdos da área de visualização, não sendo possível voltar a recuperá-los.

Correr ficheiro de *log* Será possível, dado um ficheiro de registo do DGS ou do *SOAP Adaptor*, reproduzir uma sessão de uso, tanto do DGS como do próprio *SOAP Client*, ao ler as chamadas e respectivos parâmetros do ficheiro e executá-las, mostrando o resultado de cada chamada na área de visualização.

Criar base de dados local Será possível povoar uma base de dados local (também chamada base de dados *stub*) com os dados necessários para correr o DGS, usando apenas os métodos disponibilizados pelo *SOAP Adaptor* e algumas informações que constam na base de dados de HUMS.

Fazer *logging* das chamadas Será possível activar ou desactivar o registo das chamadas efectuadas feito pelo *SOAP Adaptor*.

Validação dos parâmetros das chamadas Será possível activar e desactivar a validação feita aos parâmetros passados às chamadas. Note-se que deverão ser mantidas as validações de tipos de dados de modo a evitar excepções lançadas pela aplicação.

3.3 Resumo e Conclusões

Dado que o DGS já está estruturado de forma a que o acesso a dados esteja bastante separado do resto da funcionalidade, a solução óbvia é criar um módulo que mantenha a interface do módulo original (o *OO4O*) mas com funcionalidade diferente, ligando-se a um *Web Service* ou a uma base de dados. Para facilitar o processo de testes, dado que as operações do DGS usualmente envolvem várias chamadas ao serviço em uso, foi decidido desenvolver uma aplicação que usa a mesma biblioteca para fazer as chamadas, permitindo executar cada chamada individualmente, sendo os parâmetros fornecidos pelo utilizador. Esta aplicação permite ainda criar uma cópia local dos dados do serviço em uso e executar a sequência de comandos de uma sessão de uso do DGS, ou do próprio cliente, através de um ficheiro de *log*.

Os requisitos para o componente principal do projecto, o *SOAP Adaptor*, são relativamente reduzidos visto que é suposto esta alteração não ter qualquer impacto ao nível do funcionamento da aplicação. Os requisitos para o *SOAP Client*, devido à própria natureza da aplicação, são muito voltados para facilitar todo o processo de testes, tanto do *Web Service* como do DGS. Foi dada grande atenção aos ficheiros de registo para estes serem o mais semelhantes possível de forma a poderem ser todos lidos a partir do *SOAP Client*. Isto permite facilitar os testes e a detecção de erros no DGS (como já foi referido anteriormente), assim como automatizar os próprios testes, acabando por servir como *scripts* de teste.

Capítulo 4

Implementação

Neste capítulo são descritos, com mais profundidade que no capítulo anterior, os vários componentes da solução. São também detalhados os aspectos mais relevantes da implementação de cada componente, o ambiente de testes e o procedimento de testes.

4.1 *SOAP Adaptor*

O *SOAP Adaptor* funciona como uma biblioteca, sendo por isso desenvolvido como uma DLL. É uma biblioteca relativamente genérica, sendo usada neste projecto tanto pelo DGS (C++) como pelo *SOAP Client* (C#) e pelo *SOAP Server* (descrito na Secção 4.1.1).

Na Figura 4.1 pode-se ver o diagrama de classes simplificado do *SOAP Adaptor*. O diagrama foi simplificado porque dado que os tipos de dados têm de ser exportados para o HAL, existe uma classe para cada tipo de dados complexo (*Metric*, *Aircraft*, etc, um total de 13 classes mais os respectivos conjuntos de elementos e as respectivas interfaces) que não acrescentam grande informação ao diagrama. Desta forma simplifica-se a leitura do diagrama.

A classe *CSoapAdaptor* funciona como a base da biblioteca. É ela que expõe os vários métodos oferecidos originalmente pelo *OO4O*, assim como alguns métodos utilitários adicionais como activar/desactivar o *logging* e configurar uma ligação a um servidor *proxy* para a ligação ao *Web Service*. São disponibilizados ainda dois métodos para inicializar as fonte de dados: *initDatabase()* e *initWebService()*. Depois de inicializar uma das duas fontes de dados, as chamadas feitas aos métodos serão encaminhadas para os métodos respectivos da classe instanciada durante a inicialização.

Ao inicializar o *Web Service*, é instanciada a classe *DgsWebServiceWrapper*. Esta classe não é, contudo, o *proxy* do serviço remoto mas sim, como o nome indica, um *wrapper* para o *proxy*. Isto foi necessário devido à necessidade de poder especificar o uso

do HTTP versão 1.0, tendo sido feito *override* a um dos métodos de uma das superclasses dos *proxys* de *Web Services*, *System.Net.WebClient*. A superclasse da *DgsWebService-Wrapper* é a classe *DgsWebService*, que é gerada a partir da WSDL do serviço.

Ao inicializar a base de dados, é instanciada a classe *CFlyvisDatabase*. Esta é responsável por implementar a API do *SOAP Adaptor* de forma a ter o mesmo comportamento que o *Web Service*. Ao instanciar a classe *CFlyvisDatabase* é também instanciada a classe *CDBWrapper* que, como o nome indica, é um *wrapper* para o acesso à base de dados. Apesar do ODBC já permitir alguma abstracção em relação ao SGBD, esta não cobre certos casos como, por exemplo, a selecção do próximo valor de uma sequência. Em Ingres a selecção é feita da seguinte forma:

```
SELECT sequencia.nextval
```

Contudo, em Oracle é preciso referir a tabela *DUAL*, que é uma tabela de sistema com apenas uma linha (contendo o valor "X") usada para seleccionar constantes, pseudo-colunas e expressões[1]. A *query* toma então a seguinte forma:

```
SELECT sequencia.nextval FROM DUAL
```

Esta classe é ainda necessária para a criação das tabelas da base de dados *stub* visto haver discrepâncias nos nomes dados aos tipos de dados, nos parâmetros para definir tipos de dados e ainda na maneira como as próprias dimensões das colunas são definidas.

4.1.1 *SOAP Server*

Durante a fase inicial do desenvolvimento o serviço SAP ainda não estava finalizado, tendo sido criado um *Web Service* próprio para ir experimentando o código desenvolvido. Este, para simplificar, foi implementado recorrendo à plataforma .NET. Na sua essência, o servidor era apenas um *wrapper* para o *SOAP Adaptor*, expondo os seus métodos e usando uma ligação a uma base de dados Oracle com os dados de teste. O nome dado ao serviço foi *SOAP Server*.

4.1.2 Comunicação com o HAL

A classe *CSoapAdaptor* implementa uma interface, a *ISoapAdaptor*. Da mesma forma, todas as classes que definem os tipos de dados complexos usados pelo HAL implementam uma interface que define os seus métodos e variáveis. Isto deve-se ao facto de o HAL comunicar com o *SOAP Adaptor* através de DCOM[2], sendo portanto necessário que cada classe usada tenha uma interface disponível, cada uma delas identificada por um GUID[3].

[1] http://download.oracle.com/docs/cd/B14117_01/server.101/b10759/queries009.htm
[2] http://en.wikipedia.org/wiki/Distributed_Component_Object_Model
[3] *Globally Unique Identifier*

Existem também classes que abstraem os vectores de tipos complexos com, mais uma vez, as respectivas interfaces. Apesar de não ser estritamente necessário visto que DCOM consegue lidar com vectores normais, foi implementado desta forma visto que facilita a implementação do lado do HAL devido aos métodos que as classes disponibilizam para aceder aos elementos do vector. Estes métodos facilitam a manipulação dos dados ao diminuir consideravelmente a manipulação de apontadores dentro do HAL.

4.1.3 Integração com BEA WebLogic

Dado a natureza dos *Web Services*, à partida não iria haver grandes alterações a fazer para adaptar o código desenvolvido à plataforma do cliente. Contudo, quando o serviço SAP foi finalmente disponibilizado, surgiu um problema. Em WebLogic, os *Web Services* usam uma forma de troca de mensagens diferente da que é usada usualmente. Enquanto que normalmente é usado um sistema baseado em pedido-resposta, o WebLogic usa *Conversations*. Como o nome indica, as comunicações são feitas dentro de uma "conversa", o que na prática significa que são feitas dentro de um contexto específico. Este comportamento é análogo às sessões HTTP, mantendo o estado ao longo de uma série de comunicações. A adaptação do código .NET para ter em consideração as *Conversations* consiste em adicionar cabeçalhos extra aos pedidos SOAP, criando um identificador (*conversationID*) novo para cada sessão ou usando o identificador actual [Rod05]. Dado que o desenvolvimento roda à volta de uma WSDL, que é actualizada sempre que o ICD é alterado, é preciso também actualizá-la para suportar os novos cabeçalhos e os novos métodos para manipular esses cabeçalhos. Isto é feito através de uma pequena ferramenta fornecida pela Siemens que, dado uma WSDL, cria uma nova já com as informações adicionais.

4.1.4 *Timeouts* do *Web Service*

Os *Web Services* implementados num servidor WebLogic têm um mecanismo de *timeout* baseado em dois valores [Rod05]:

Maximum Idle Time Tempo máximo que uma *Conversation* pode estar inactiva;

Maximum Age Tempo máximo que uma *Conversation* pode durar.

Quando o *Web Service* envia uma excepção de *timeout*, a detecção é feita no *SOAP Adaptor*, sendo lançada uma excepção adequada. Lidar com essa excepção no código C# é trivial, bastando apanhar a excepção num bloco *try*/*catch*. Contudo, revelou-se mais complicado fazê-lo no código C++ do HAL. Dado que este não conhece os tipos de dados das excepções da plataforma .NET, a única coisa que surge quando a excepção é lançada pelo *SOAP Adaptor* é uma excepção genérica e vazia, não sendo possível identificá-la.

A solução para este problema foi adicionar ao *SOAP Adaptor* um código de erro que é verificado quando se apanha uma excepção, indicando assim se ocorreu um *timeout*.

4.1.5 Definições Regionais

Dado que a aplicação é usada pela RDAF, tem de estar preparada para as definições regionais usadas na Dinamarca. À primeira vista parecia ser algo de somenos importância mas existem algumas diferenças consideráveis para os *standards* a que se está habituado. Por exemplo, o formato usado para as datas é *dd.mm.yyyy* (dia.mês.ano), usando pontos para separar os valores. Contudo, a generalidade dos SGBDs favorece datas separadas por hífenes. Isto deu alguns problemas visto que apesar de Oracle ter funções para converter datas entre diferentes formatos, Ingres, surpreendentemente, não tem, forçando a fazer a conversão para um formato que o SGBD entenda do lado do código C#. Outra grande diferença é o separador decimal. O separador usado na Dinamarca é a vírgula, e não o ponto como é vulgarmente usado (apesar de tecnicamente o separador usado em Portugal ser também a vírgula). Isto tem implicações óbvias a nível do código SQL, como se pode ver a seguir:

```
INSERT INTO tabela (inteiro, decimal) VALUES (1, 3.2)

INSERT INTO tabela (inteiro, decimal) VALUES (1, 3,2)
```

A segunda *query* vai obviamente dar origem a um erro visto que o número decimal é interpretado como se fossem dois parâmetros, tentando assim inserir três valores em duas colunas. Ambos os problemas foram resolvidos através de um processamento cuidadoso de todos os valores que são passados como parâmetros, verificando sempre se se encontra no formato correcto e especificando uma definição cultural invariante ao fazer a conversão de números e datas para texto e vice-versa. O correcto funcionamento foi verificado tendo o *SOAP Server* a correr numa máquina com as definições regionais americanas e estando o cliente, neste caso o *SOAP Client*, a correr numa máquina com as definições regionais dinamarquesas.

4.2 *SOAP Client*

Como já foi referido, o *SOAP Client* foi desenvolvido usando o Visual Studio da Microsoft. Isto simplificou bastante a desenho da interface gráfica dado a facilidade de uso do seu *interface designer*. A Figura 4.3 permite ter uma ideia do aspecto da aplicação. Quando a aplicação é iniciada, é lido um ficheiro de configuração com as várias fontes de dados disponíveis com o mesmo formato do que é usado pelo DGS. Isto evita a escrita de URLs e de nomes de ligações ODBC, facilitando o papel primário da ferramenta, que é ajudar a corrigir problemas no funcionamento do DGS, tentando recriar o seu ambiente.

Algumas das operações, nomeadamente o *Stub Creator* e correr um ficheiro de *log*, são algo demoradas, levando a que a interface gráfica fique bloqueada à espera que estas terminassem. A solução óbvia foi fazer essas operações num *thread* separado. Inicialmente foi usada a classe *BackgroundWorker*, que abstrai para métodos e eventos a maioria das funcionalidades com que se lida em tarefas do género, como por exemplo eventos que são lançados quando o *thread* termina e eventos e métodos para enviar algum tipo de medida de progresso para o *thread* pai. Apesar desta abordagem se revelar bastante confortável, tem um problema que a inviabilizou: o cancelamento de um *thread* que esteja a correr. Apesar da classe ter um método para este efeito, este limita-se a definir a propriedade *CancellationPending* como *true* e o código que está a ser executado no *thread* é que tem a responsabilidade de verificar essa propriedade periodicamente e terminar case esteja activada. Como parte do código a ser corrido se encontra numa biblioteca separada, não fazia sentido estar a verificar uma determinada propriedade completamente externa. Por esta razão optou-se por simplesmente usar a classe *Thread* que, ao contrário da *BackgroundWorker*, implementa uma versão mais básica de um *thread*. A classe *Thread* contém o método *Abort()* que lança uma excepção dentro do *thread*, que depois pode ser tratada adequadamente.

O uso de *threads* simples criou um problema: em .NET a actualização dos controlos da interface gráfica só pode ser feita através do *thread* que as criou. Enquanto que a classe *BackgroundWorker* prevê isso com o seu método e evento para reportar progresso, a classe *Thread* não tem essa funcionalidade de base. A solução foi acrescentar às classes relevantes um *delegate* para uma função que recebe como parâmetro o texto que contém a actualização em termos e progresso. No caso do *SOAP Client* esse *delegate* é definido como uma função que actualiza a área de visualização com o estado actual.

4.2.1 Chamar Métodos do Serviço

Esta funcionalidade foi relativamente simples de implementar, tendo apenas sido algo trabalhoso implementar e testar as validações dos parâmetros inseridos pelo utilizador, em parte devido às questões relacionadas com as definições regionais referidas na Secção 4.1.5. Foi criada uma *tab* para cada chamada disponibilizada pelo serviço. Estas estão agrupadas por cores de acordo com as suas funções (controlo da sessão, *debrief*, engenharia, etc). O método actual é destacado com uma cor diferente para ser mais fácil identificá-lo visualmente. Dado que alguns métodos têm um grande número de parâmetros, em que alguns são valores textuais com um formato específico, foi implementado, quando os parâmetros eram numerosos, um botão que preenche os campos com valores semi-aleatórios.

Na Figura 4.4 pode ver-se o ecrã de chamada de métodos em que foram feitas duas chamadas: uma para ir buscar todos os helicópteros e uma segunda para ir buscar um

helicóptero específico. Na área de visualização à direita podem-se ver as chamadas com os respectivos parâmetros e os seus resultados.

Os métodos associados aos botões de chamada limitam-se a processar os parâmetros introduzidos e a fazer as devidas validações (se estiverem activadas). Os parâmetros são passados para um segundo método responsável por fazer a chamada ao *SOAP Adaptor* e mostrar os resultados. Isto permite que esse método seja reutilizado pela funcionalidade de correr ficheiros de *log* (descrita na Secção 4.2.2), sendo necessário apenas retirar os vários parâmetros de cada chamada e fazer a chamada.

4.2.2 Correr Ficheiros de *Log*

A implementação da funcionalidade de correr um ficheiro de *log* (Figura 4.5) foi relativamente simples visto o *log* ter um formato bem definido:

```
método -> parâmetro1:valor1, parâmetro2:valor2
```

O processamento do ficheiro é feito lendo-o linha a linha, recorrendo a expressões regulares para identificar as chamadas e os parâmetros. Depois de extrair a informação, usa o mesmo método que as chamadas feitas através do *SOAP Client* para fazer a chamada e apresentar os resultados. Sempre que uma chamada efectua alterações na fonte de dados actual, é apresentada uma caixa de diálogo para confirmar a alteração.

4.2.3 *Stub Creator*

O *Stub Creator* tem como limitação poder usar apenas os métodos disponibilizados pelo *Web Service*, pelo que foi necessário analisar o esquema da base de dados *stub* assim como os parâmetros e valores de retorno de cada método do *Web Service*. A partir dessa análise, extrai-se a sequência de chamada dos vários métodos e os valores a reter para usar nas chamadas seguintes. Na Figura 4.6 pode-se ver o diagrama de sequência que descreve este processo. A base de dados onde vai ser criada a cópia local é designada por *Target DB*. Note-se que ao contrário das interacções com a base de dados como fonte de dados para executar as chamadas normais (através do *SOAP Adaptor*), as inserções na *Target DB* são feitas manipulando directamente a base de dados.

É de referir que a base de dados criada não fica idêntica nem provém toda da fonte de dados original visto existirem informações que não são acessíveis através das chamadas disponibilizadas. Estas informações são os utilizadores do sistema e as respectivas palavras-passe e a informação relativa aos voos de cada helicóptero. Relativamente aos voos, foi possível dar a volta à situação recorrendo à base de dados de HUMS, que contém forçosamente todos os voos feitos pelos helicópteros monitorizados. A partir de cada identificador de voo já é possível retirar as restantes informações necessárias que dependem de chamadas que têm esse identificador como um dos seus parâmetros. Quanto

aos utilizadores e palavras-passe, faz todo o sentido que essa informação não esteja disponível. Caso contrário, seria uma grave falha de segurança visto que estaria a disponibilizar dados sensíveis. Como a base de dados *stub* vai ser usada apenas localmente, o compromisso arranjado foi, para cada *role* no sistema, criar um utilizador com nome de utilizador e palavra-passe iguais ao nome desse *role*. Isto permite que o criador da base de dados tenha acesso ao DGS, podendo depois criar utilizadores e alterar ou apagar os utilizadores existentes.

Na Figura 4.7 podemos ver o ecrã do *SOAP Client* a partir de onde se pode correr o *Stub Creator*, podendo-se também criar as tabelas necessárias na base de dados alvo, caso elas não existam.

Na Figura 4.8 é apresentado o diagrama físico da base de dados *stub*. O *schema* foi herdado dos autores originais do DGS, não podendo ser alterado para manter a compatibilidade com outras ferramentas existentes. A estrutura não é a melhor, existindo algumas colunas desnecessárias e alguma redundância de informação. Por exemplo, as colunas *part_number*, *serial_number* e *kokomp_id*, que juntas formam uma combinação única, estão presentes nas tabelas *stub_asset_metric*, *stub_asset_details* e *stub_asset_metric_details*, o que toma particular relevância tendo em conta que a tabela *stub_asset_metric_details*, povoada com os dados fornecidos pela RDAF, tem para cima de 250.000 registos.

Inicialmente, a sequência de operações foi baseada numa ferramenta desenvolvida em Microsoft Access, com VisualBasic, que também tinha o propósito de criar uma base de dados *stub*, mas com uma grande diferença: esta ferramenta acede directamente à base de dados do MMS da RDAF e não através dos métodos, na altura, disponibilizados pelo *OO4O*. Devido a isso, a implementação inicial do *Stub Creator* era pouco eficiente, demorando perto de 1 hora a criar a base de dados local. Cerca de 90% desse tempo era gasto ao carregar a tabela *stub_asset_metric_details* que, como já foi referido, tem um número muito grande de registos. Após algumas pequenas optimizações com impacto negligenciável, foi dado conta que, também ao carregar a tabela *stub_asset_metric_details*, havia duas operações que podiam ser aglutinadas numa só sendo necessário um número reduzido de alterações nas classes dos tipos de dados. Esta alteração teve um efeito dramático, tendo o tempo de execução diminuído de aproximadamente 60 minutos para apenas 10 minutos.

O principal problema que surgiu durante os testes informais/desenvolvimento foi ao carregar a tabela *stub_asset_metric_details*. A pesquisa sobre a mensagem de erro apresentada não foi muito elucidativa visto indicar apenas que a mensagem era sintoma de um erro vindo do *driver* ODBC de Ingres. Como se trata de código não-*managed*, não é possível fazer *debug* para detectar a causa. Após bastante pesquisa infrutífera foi experimentado um *driver* ODBC alternativo, não tendo os resultados sido muito diferentes. O erro simplesmente demorava mais tempo a surgir. Ao experimentar criar a base de dados local em Oracle, surgiu um erro mais explícito: não havia mais cursores livres.

Após alguma investigação, revelou-se que o erro provinha do facto do *garbage collector* de .NET não ser previsível, ou seja, correr quando a plataforma entende ser mais apropriado [Dor07]. Dado que o driver não liberta os cursores automaticamente, isto estava a levar a que o objecto da ligação à base de dados, ao não ser destruído, não libertasse os cursores usados. Isto foi resolvido utilizando o método *Dispose()*, que força a plataforma a libertar os recursos em uso pelo objecto. Apesar de este ser o comportamento habitual dos *garbage collectors* [JL96], o problema não surgiu mais cedo visto que na fase anterior do processo de povoar a base de dados *stub* a quantidade de dados a inserir é consideravelmente menor, dando mais tempo ao *garbage collector* para executar.

4.3 Testes

Como foi referido anteriormente, inicialmente os testes foram feitos através de um *Web Service* próprio, baseado no *SOAP Adaptor*. Isto permitiu testar, ao mesmo tempo, tanto a interacção com o *Web Service* como com a base de dados local. A Figura 4.9 demonstra o esquema de testes criado com o *SOAP Server*. Pode ver-se que tanto a aplicação de HUMS (o DGS) como o *SOAP Client* se podem ligar a qualquer um dos três serviços que estavam disponíveis O esquema contempla também a funcionalidade do *SOAP Adaptor* de criar uma base de dados *stub*, daí a sua ligação à *Stub DB*.

Informalmente, os testes realizados foram feitos à medida que se ia desenvolvendo, utilizando principalmente o *SOAP Client*. Dado que a fonte de dados estava disponível foi possível verificar, quando necessário, que o resultado das chamadas estava correcto. No fim da fase de desenvolvimento testou-se principalmente através do DGS, efectuando *debriefs*, que é a funcionalidade que exercita mais o *SOAP Adaptor*.

Inicialmente, os testes ao *Stub Creator* revelaram-se problemáticos porque era necessário comparar os conteúdos de duas bases de dados que podiam ser de tipos diferentes (Ingres e Oracle). Após alguma pesquisa e vários programas que por razões várias não eram adequados, foi usado o CompareData[4]. Este programa usa drivers ODBC, funcionando por isso com uma enorme gama de SGBDs, incluindo Ingres e Oracle.

A nível formal, os testes realizados foram os testes de aceitação do DGS 1.2, dado que o principal requisito é não alterar a funcionalidade do DGS. Estes testes baseiam-se num conjunto de procedimentos, cada um com uma sequência de passos e o resultado esperado. Estes utilizam como fonte de dados cartões de memória virtuais, que simulam os cartões de memória do próprio helicóptero. Estes cartões virtuais são específicos para cada teste, sendo referido no procedimento de teste quais os cartões a usar.

[4] http://www.zidsoft.com

4.4 Resumo e Conclusões

O *SOAP Adaptor* é uma biblioteca que vai ter vários casos de uso, seja com o DGS ou com o *SOAP Client*, ou até ferramentas futuras, tendo portanto que ser robusta e manter um comportamento coerente em vários ambientes diferentes e ao lidar com fontes de dados diferentes. A alteração que foi necessária para o *SOAP Adaptor* conseguir comunicar com o servidor WebLogic, apesar de não ter sido complexa, desapontou um pouco visto ter quebrado com o princípio dos *Web Services* serem independentes de plataforma. É algo paradoxal existir um documento [Rod05] a falar de interoperabilidade de uma tecnologia que é suposto ser ela própria um facilitador de interoperabilidade.

O *SOAP Client* é uma aplicação típica desenvolvida em *Windows Forms*, fazendo uso do modelo *event-driven* do .NET e usando o *SOAP Adaptor* como a biblioteca para acesso a dados. O seu desenvolvimento, em particular do *Stub Creator*, foi uma experiência interessante visto enveredar pela área da optimização. Os resultados atingidos foram bons, sendo a acção executada num espaço de tempo relativamente curto.

Figura 4.1: Diagrama de Classes Simplificado do *SOAP Adaptor*

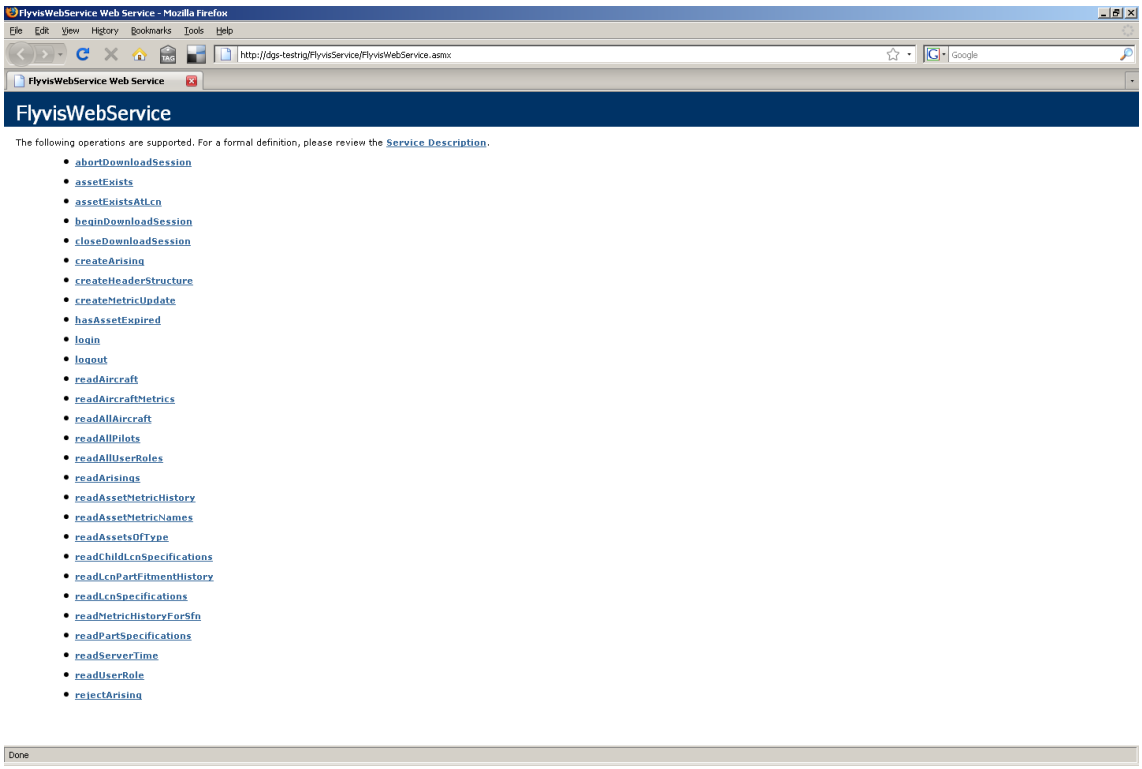

Figura 4.2: Interface *Web* do *SOAP Server*

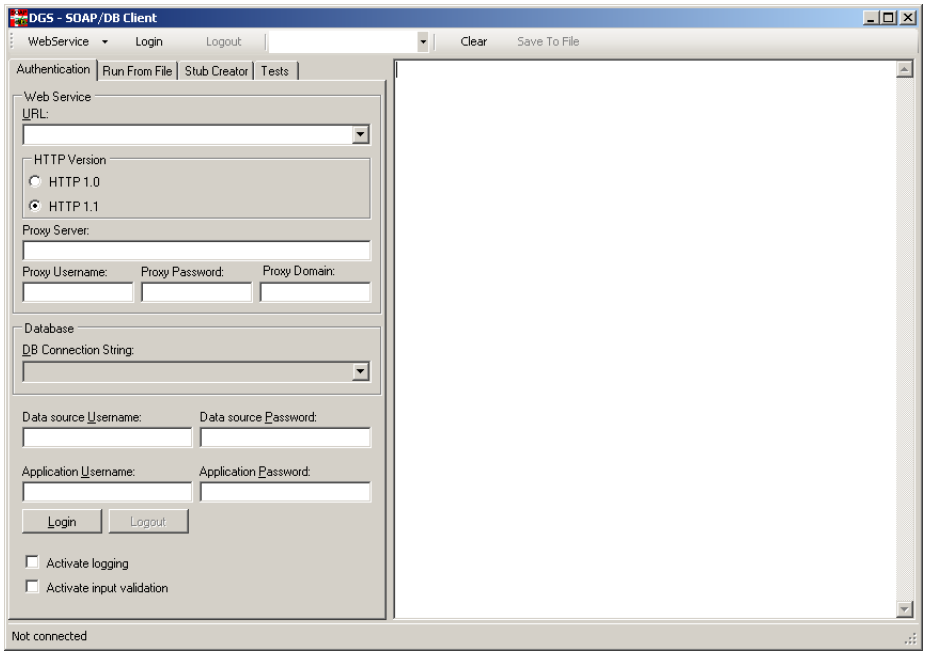

Figura 4.3: Ecrã de Autenticação do *SOAP Client*

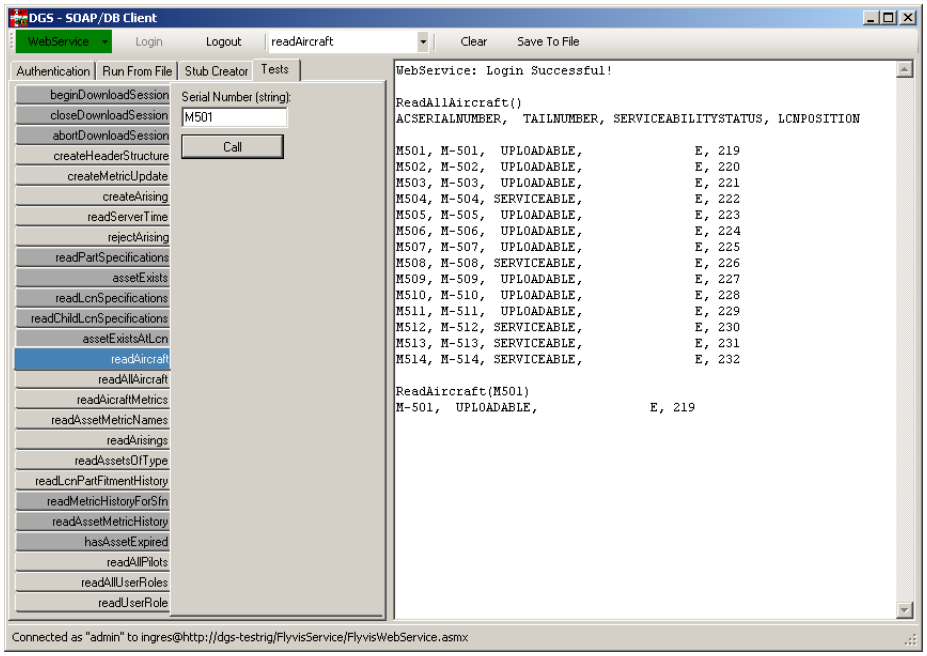

Figura 4.4: Ecrã do *SOAP Client* para Chamar os Métodos do Serviço

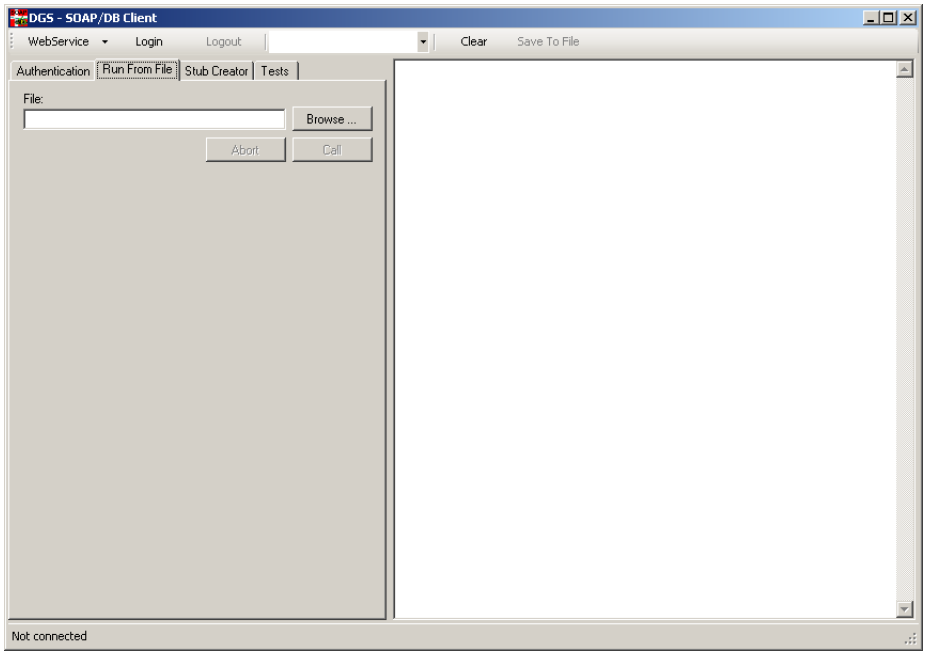

Figura 4.5: Ecrã do *SOAP Client* para Correr Ficheiro de *Log*

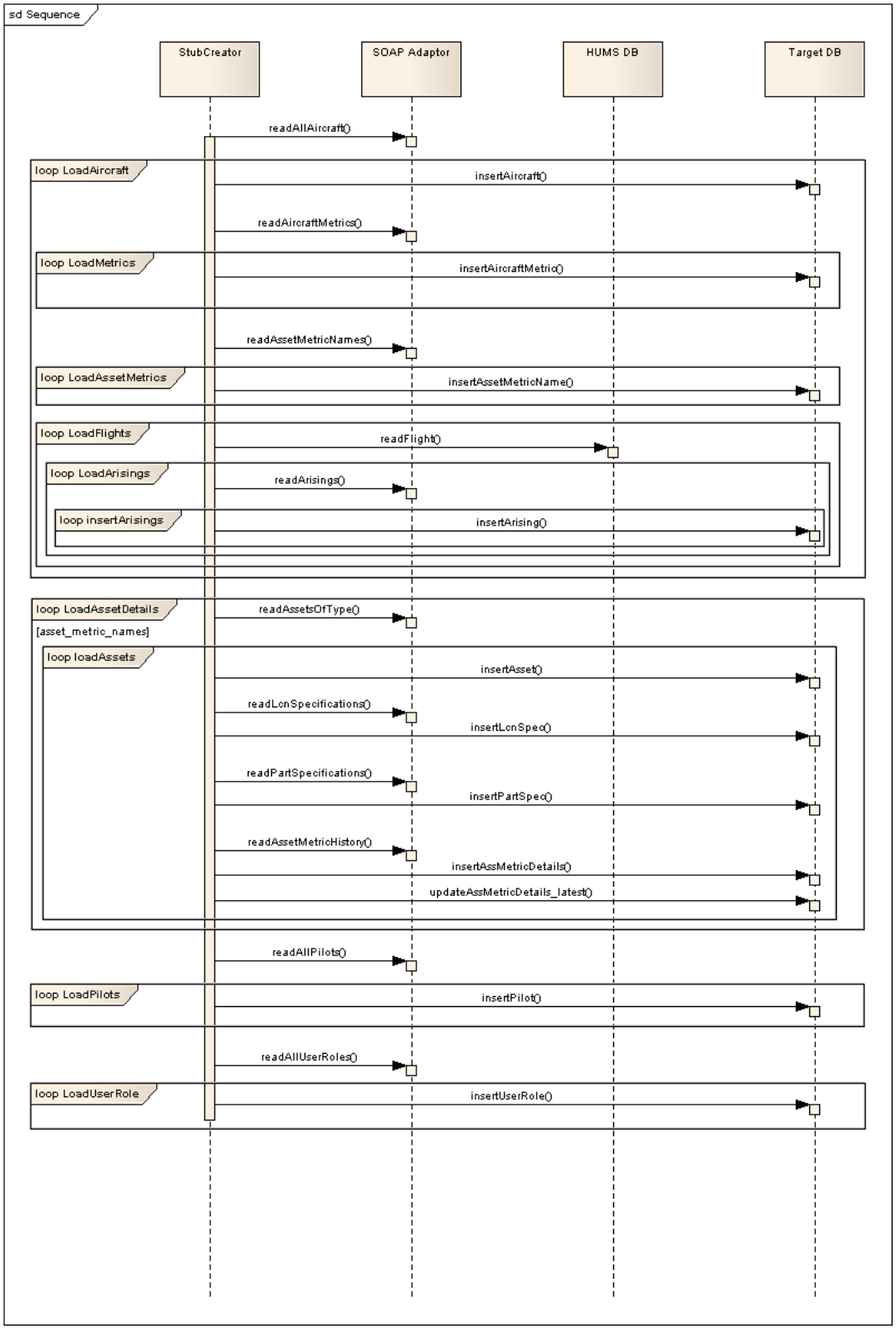

Figura 4.6: Diagrama de Sequência do *Stub Creator*

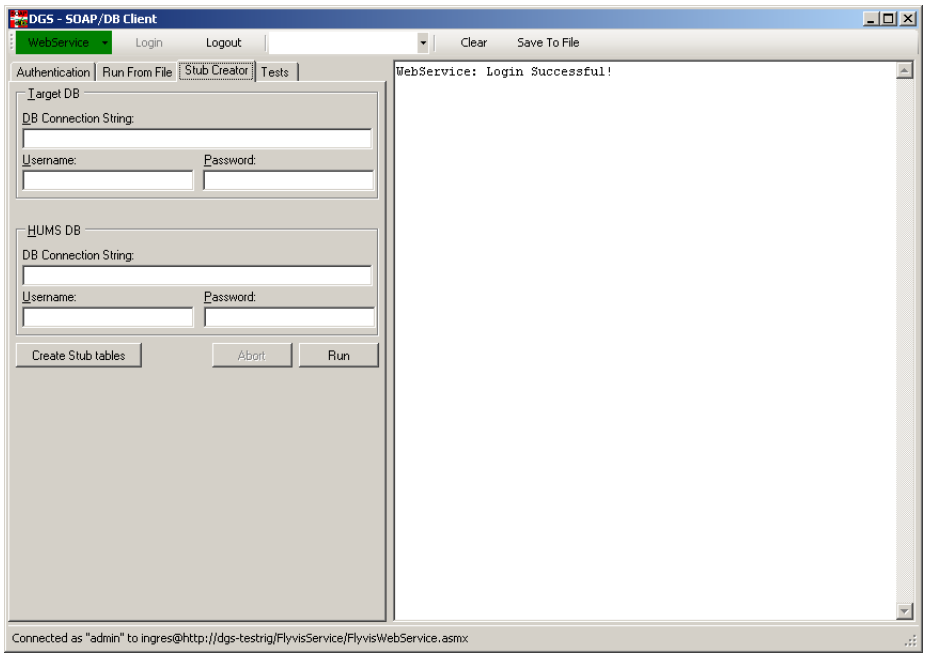

Figura 4.7: Ecrã do *SOAP Client* com o *Stub Creator*

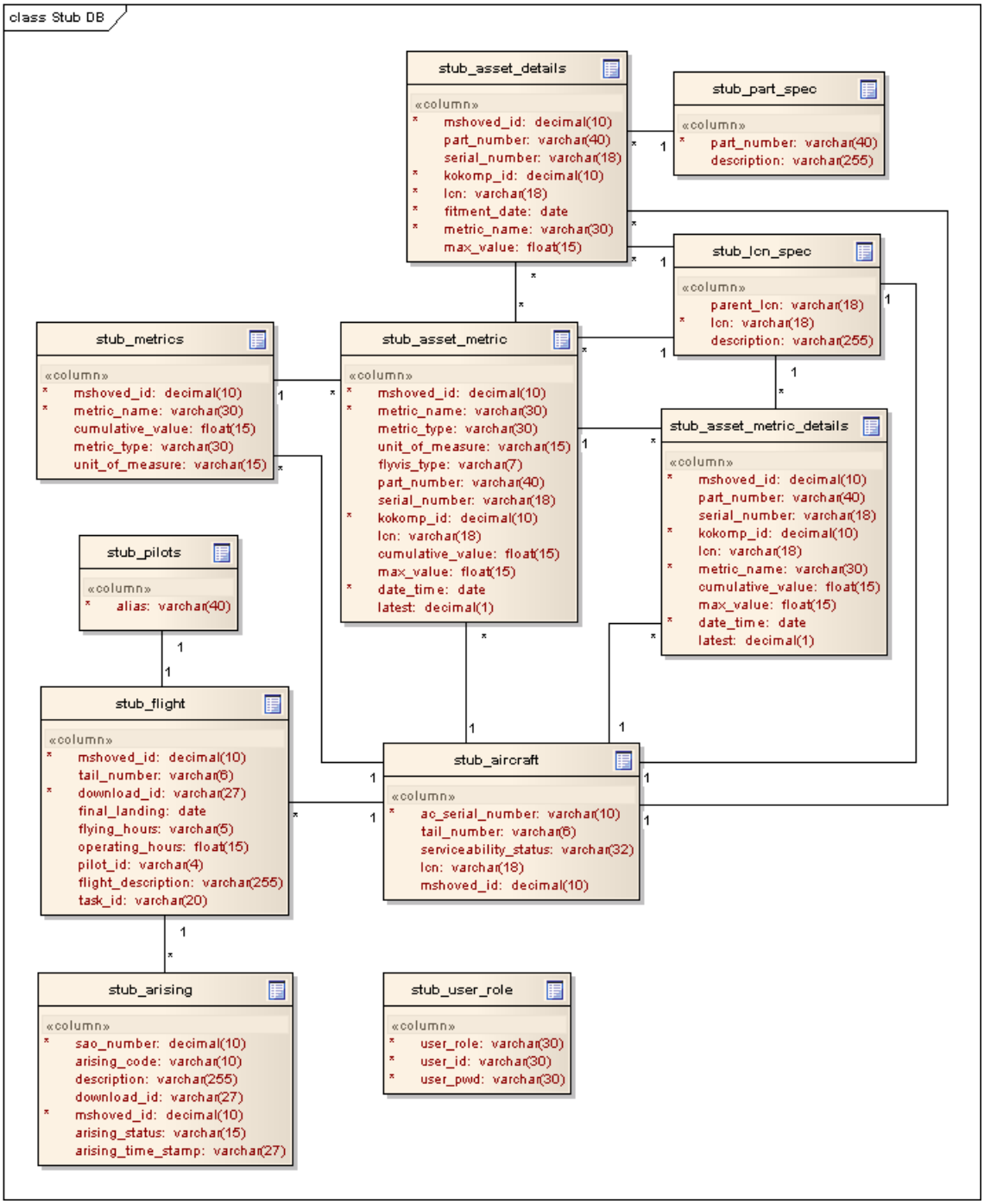

Figura 4.8: Diagrama Físico da Base de Dados de *Stub*

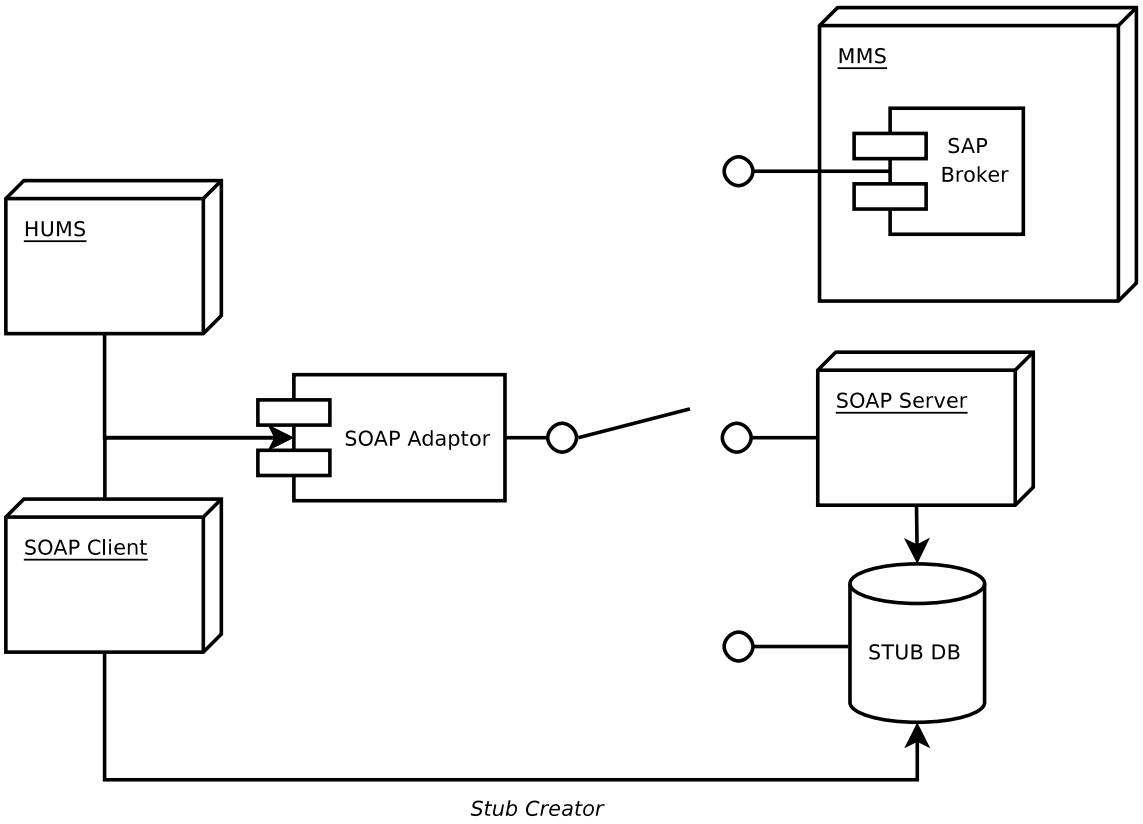

Figura 4.9: Esquema de Testes

Capítulo 5

Conclusões e Trabalho Futuro

O projecto decorreu como planeado, tendo havido apenas um ligeiro atraso no início da fase de testes. As funcionalidades planeadas foram todas implementadas, não tendo havido grandes sobressaltos. Os únicos problemas que surgiram foram relacionados com recursos da base de dados não estarem a ser libertados devido à não-previsibilidade do *garbage collector* de .NET e com alguns aspectos mais intrincados do uso de *threads*.

Este projecto é mais uma demonstração de que *Web Services* são uma excelente forma de integrar diferentes sistemas. Estes permitem, graças ao uso de tecnologias simples, maduras e muito disseminadas (como o HTTP), superar dificuldades e quebrar barreiras à adopção com que outras tecnologias como CORBA, DCOM e RMI se debateram. Contudo, para usar as palavras de Fred Brooks num contexto análogo, os *Web Services* não são a *silver bullet*[1] para o problema da interoperabilidade, como se pode ver pelo exemplo de um cliente de *Web Services* (neste caso .NET, mas poderia suceder com qualquer outra linguagem/plataforma) que precisa de ser adaptado para comunicar com um servidor de *Web Services* que, neste caso, até é bastante popular.

5.1 Satisfação dos Objectivos

Os objectivos do projecto foram satisfeitos em pleno. Relativamente ao DGS e ao *SOAP Adaptor*, o principal objectivo, que era não alterar a funcionalidade do DGS, foi cumprido, tendo este passado todos os testes a que foi submetido, com os clientes, nas instalações da AgustaWestland. Em termos de desempenho, não houve uma diferença relevante no tempo total de execução de um *debrief*, que é o processo mais moroso e mais intensivo em termos de acesso a dados.

[1] http://www.lips.utexas.edu/ee382c-15005/Readings/Readings1/05-Broo87.pdf

Relativamente ao *SOAP Client* os objectivos foram claramente superados. A ferramenta acabou por ser mais útil do que se esperava, tendo inclusive a equipa da Siemens pedido acesso à ferramenta para melhor testar o seu próprio *Web Service*. A criação da base de dados local a partir do serviço a que se está ligado também foi de grande utilidade para replicar e converter bases de dados entre SGBDs diferentes. No caso inicial, a base de dados era fornecida pela RDAF para Oracle, sendo depois importada para Ingres.

5.2 Trabalho Futuro

Um trabalho futuro seria fazer uma comparação mais detalhada entre SOAP e REST, implementando o *Web Service* com REST e adaptando o *SOAP Adaptor* adequadamente. Desta forma seria possível extrair métricas de ambas as implementações confirmando, ou não, as conclusões retiradas do Capítulo 2. Contudo, devido às razões enunciadas, existe alguma certeza que as conclusões serão confirmadas e não desmentidas.

Um melhoramento a nível de desenho que se poderia fazer seria redesenhar a base de dados *stub*, tornando-a mais bem estruturada, o que simplificaria algumas das operações e traria certamente uma melhoria de desempenho. Um exemplo claro baseia-se no que já foi referido na Secção 4.2.3. Se se juntassem as três colunas numa única tabela com um identificador comum, uma pesquisa por essas mesmas colunas nas tabelas em causa, sendo uma delas de longe a tabela com mais registos, seria necessário apenas comparar uma coluna em vez de três. A isto acrescenta-se, obviamente, outras colunas pelas quais se queira filtrar a informação. Seria, contudo, necessário actualizar as ferramentas existentes que dependem do *schema* actual.

Outro trabalho futuro que poderá ter algum interesse, se bem que fora do âmbito deste projecto em particular, seria aplicar processos e técnicas de *data mining* aos dados recolhidos pelos sensores de bordo e tratados pela aplicação de HUMS. Por exemplo, informação sobre falhas e avarias poderia ser usada para detectar eventuais padrões nas falhas, tais como determinadas sequências de acções, factores ambientais, etc.

Referências

[Asa08] Amit Asaravala. Giving SOAP a REST, 2008. Disponível em http://www.devx.com/DevX/Article/8155, acedido a última vez em 11 de Junho de 2008.

[Des05] SAP News Desk. Microsoft, IBM, SAP To Discontinue UDDI Web Services Registry Effort @ SOA WORLD MAGAZINE, Dezembro 2005. Disponível em http://webservices.sys-con.com/read/164624.htm, acedido a última vez em 11 de Junho de 2008.

[Dor07] Scott Dorman. Using Garbage Collection in .NET, Julho 2007. Disponível em http://geekswithblogs.net/sdorman/archive/2007/07/21/Using-Garbage-Collection-in-.NET.aspx, acedido a última vez em 25 de Junho de 2008.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach e T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

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

[Gro04] Web Services Architecture Working Group. Web Services Architecture. Technical report, World Wide Web Consortium, 2004. Versão mais recente à data disponível em http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[Gro06] SOA Working Group. Definition of SOA. Technical report, The Open Group, 2006. Disponível em http://www.opengroup.org/projects/soa/doc.tpl?gdid=10632.

[JL96] Richard E. Jones e Rafael Dueire Lins. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. John Wiley, 1996.

[Rod05] Jesús Rodríguez. Interoperability with Message Exchange Patterns Created Using BEA WebLogic 8.1.3, Janeiro 2005. Disponível em http://msdn.microsoft.com/en-us/library/ms978494.aspx, acedido a última vez em 30 de Abril de 2008.

[W3C07] W3C XML Protocol Working Group. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Technical report, World Wide Web Consortium, 2007. Versão mais recente à data disponível em http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[Wik08a] Wikipedia, the free encyclopedia. Representational State Transfer, 2008. Disponível em http://en.wikipedia.org/wiki/Representational_State_Transfer, acedido a última vez em 11 de Junho de 2008.

[Wik08b] Wikipedia, the free encyclopedia. Service-oriented architecture, 2008. Disponível em http://en.wikipedia.org/wiki/Service_Oriented_Architecture, acedido a última vez em 11 de Junho de 2008.

[Wik08c] Wikipedia, the free encyclopedia. Web service, 2008. Disponível em http://en.wikipedia.org/wiki/Web_Service, acedido a última vez em 11 de Junho de 2008.