UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

IMPLEMENTAÇÃO DE UM SERVICE BROKER

Filipe Miguel da Silva Rodrigues

MESTRADO EM ENGENHARIA INFORMÁTICA

Especialização em Arquitectura, Sistemas e Redes de Computadores

2010

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

IMPLEMENTAÇÃO DE UM SERVICE BROKER

Filipe Miguel da Silva Rodrigues

PROJECTO

Trabalho orientado pelo Prof. Doutor Miguel Nuno Dias Alves Pupo Correia

e co-orientado pelo Eng.º Ricardo Manuel Atanásio Lopes Beja

MESTRADO EM ENGENHARIA INFORMÁTICA

Especialização em Arquitectura, Sistemas e Redes de Computadores

2010

Agradecimentos

Começo por agradecer aos meus pais a oportunidade que me deram de poder tirar um curso superior. O seu apoio incondicional durante a frequência deste curso foi indispensável para a realização de um sonho.

Agradeço ao Eng.º Ricardo Beja, pela sua orientação, paciência e transmissão de conhecimentos, que durante estes nove meses me permitiram um crescimento tanto profissional como pessoal, e pela oportunidade e confiança em mim depositada.

Agradeço ao Professor Miguel Correia, pela sua disponibilidade e por todos os conselhos transmitidos que permitiram levar a bom porto a realização deste projecto.

Um agradecimento especial aos meus amigos pelo seu apoio, amizade, e palavras de incentivo.

Por fim, gostaria de agradecer aos meus colegas de trabalho, que, embora não estando directamente relacionados com a realização deste projecto, sempre se mostraram disponíveis a ajudar.

Para vocês pai e mãe.

Resumo

A implementação de um Service Broker surgiu no contexto da criação de uma nova Framework de desenvolvimento modular por parte da empresa onde este projecto foi realizado, a AMBISIG (Ambiente e Sistemas de Informação Geográfica, S.A.). Esta nova Framework tem o intuito de permitir a criação de soluções e aplicações que correspondam às necessidades do mercado global.

Com este Service Broker pretende-se criar uma separação entre a camada lógica de negócio e a camada de apresentação, de forma a garantir acessos controlados aos serviços já existentes.

De uma forma geral este componente permite implementar uma camada de controlo de acesso sobre web services registados; permite expôr *assemblies* de biblioteca como web services através da geração personalizada de um WSDL (*Web Service Description Language*) para cada *assembly*, identificando as operações possíveis de executar recorrendo à técnica de reflexão (*reflection)*, sendo posteriormente adicionada a mesma camada de controlo de acesso disponível para os web services; e, recorrendo a um portal de acesso acessível tanto a administradores como a utilizadores normais, permite que um administrador possa gerir os serviços disponibilizados pelo Service Broker, permite a configuração de permissões de invocação ao nível do serviço e/ou operações do mesmo e garante a possibilidade de monitorização de todos os pedidos realizados e registados pelo Service Broker; um utilizador normal pode consultar os serviços a que tem acesso e testar as suas funcionalidades.

Durante a realização deste projecto muitos conceitos foram testados e colocados à prova. A solução obtida embora limitada em alguns aspectos, apresenta-se funcional e com registos de desempenho satisfatórios tendo em conta os vários conceitos exploratórios em que se baseou.

Palavras-chave: SOAP, WSDL, Web Services, Reflexão, Dinamismo

Abstract

The implementation of a Service Broker arose in the context of creating a new framework for modular development of the firm where this project was conducted, AMBISIG (Environment and Geographic Information Systems, SA). This new framework aims to enable the creation of applications and solutions that meet global market needs.

This Service Broker intends to create a separation between business logic layer and presentation layer, so as to ensure controlled access to existing services.

In general this component allows to implement an access control layer in registered web services; library assemblies can be exposed as web services by generating a custom WSDL (*Web Service Description Language*) for each assembly, identifying possible operations using the reflection technique, and subsequently the same access control layer available for web services is added, and, using a portal accessible to both administrators and ordinary users, it allows an administrator to manage the services provided by the Service Broker, it allows the configuration of permissions for invoking the service and/or its operations and guarantees the possibility of tracking all requests made and recorded by the Service Broker, a normal user can check the services with as access to and he can test its functionalities.

During this project, many concepts have been tested and implemented in the prototype. The resulting solution while limited in some areas is functional and its performance is satisfactory considering the various exploratory concepts on which it relied.

Keywords: SOAP, WSDL, Web Services, Reflection, Dynamism

Notação e Glossário

ASSEMBLY Na Framework .NET da Microsoft uma assembly é uma biblioteca com código parcialmente compilado para ser usado nos lançamentos para produção, versionamento e segurança. Existem dois tipos: assemblies processuais (EXE) e assemblies de biblioteca (DLL). Uma assembly processual representa o processo que irá usar classes definidas nas assemblies de biblioteca. [6]

DI Departamento de Informática

FCUL Faculdade de Ciências da Universidade de Lisboa

HTTP Hypertext Transfer Protocol

Protocolo de comunicação utilizado para sistemas de informação de hipermédia distribuídos e colaborativos. O seu uso para a obtenção de recursos interligados levou ao estabelecimento da World Wide Web. Normalmente este protocolo faz uso do porto 80 e usa a linguagem HTML na comunicação de sítios web. [7]

MEI Mestrado de Engenharia Informática

MSDN Microsoft Development

Providencia ajuda aos desenvolvedores de software na escrita de aplicações que fazem uso dos produtos e tecnologias Microsoft. [16]

MSSQL Microsoft Structured Query Language

Linguagem de pesquisa declarativa para bases de dados relacional, sendo muitas das suas características originais inspiradas na álgebra relacional.

PEI Projecto de Engenharia Informática

PROXY Por definição, os web services usam uma rede de comunicação e protocolos standard para a troca de mensagens. Isto é, um cliente e um web service comunicam através de mensagens SOAP, que encapsulam os parâmetros de entrada e saída como XML. Uma proxy, é responsável

pelo trabalho de mapeamento de parâmetros para elementos XML e pelo envio das mensagens SOAP pela rede. Esta proxy é gerada com base na descrição do serviço (WSDL). [1]

SOA Service Oriented Arquitecture

Pode ser traduzido como *Arquitectura Orientada a Serviços* e é um estilo de arquitectura de software cujo princípio fundamental dita que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços. Frequentemente estes serviços disponibilizam interfaces, ou contratos, acessíveis através de web services ou outra forma de comunicação entre aplicações. A arquitectura SOA é baseada nos princípios da computação distribuída e utiliza o paradigma *request/reply* para estabelecer a comunicação entre os sistemas clientes e os sistemas que implementam os serviços. [9]

SOAP Simple Object Access Protocol

Protocolo projectado para invocar aplicações remotas através de RPC (*Remote Procedure Calls* – Chamadas de Procedimento Remotas) ou trocas de mensagens, num ambiente independente de plataforma e linguagem de programação. SOAP é portanto um padrão normalmente aceite para ser usado com web services. Garantindo desta forma a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização de uma linguagem (XML) e um mecanismo de transporte (HTTP) padrões. [13]

TI Tecnologias de Informação

WCF Windows Communication Foundation

Modelo de programação unificado e ambiente de execução criado pela Microsoft que visam a construção de aplicações orientadas a serviços que comunicam na web.

Usando WCF é possível enviar dados como mensagens assíncronas de um terminal para outro.

De uma forma geral, o uso de WCF permite dar uma maior flexibilidade às aplicações quanto à sua forma de comunicação, não as

limitando apenas a uma comunicação HTTP+XML (SOAP) mas estendendo essa comunicação ao TCP, UDP, entre outros. [8]

WSDL Web Services Description Language

Linguagem baseada em XML utilizada para descrever web services funcionando como um contrato do serviço. Trata-se de um documento escrito em XML que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis. [11]

XML eXtensible Markup Language

É uma recomendação do *World Wide Web Consortium* (W3C) para gerar linguagens de marcação para necessidades especiais. Este é capaz de descrever diversos tipos de dados, sendo a sua função principal de facilitar a partilha de informação através da internet. [12]

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Enquadramento

O último ano do MEI da FCUL, resume-se à frequência da cadeira de PEI. Cadeira essa que consiste na realização de um projecto de informática numa instituição externa (empresa privada ou pública), ou num dos laboratórios de investigação do DI da Faculdade.

Este documento pretende dar a conhecer todo o trabalho realizado no âmbito dessa mesma cadeira, bem como a instituição onde este foi elaborado.

O projecto foi desenvolvido numa instituição externa privada, de seu nome AMBISIG (Ambiente e Sistemas de Informação Geográfica. S.A.), com sede em Convento de S. Miguel das Gaeiras, Óbidos e escritórios na Av. Infante Santo, Lisboa. Este focou-se essencialmente na implementação de um Service Broker que consistiu no desenvolvimento de uma camada de controlo e segurança a existir entre a camada de apresentação e a camada de lógica de negócio.

A equipa de desenvolvimento que ficou responsável pela implementação deste projecto foi constituída pelo aluno, que se encontrava a realizar a referida cadeira de PEI, e o seu Co-Orientador.

1.2 Motivação

Tendo em conta as mais recentes evoluções na área das TI e a constante procura no mercado de soluções mais abrangentes, quer ao nível de integração como ao nível de plataformas, a AMBISIG pretende conceptualizar e desenvolver uma Framework de

desenvolvimento modular que permita a criação, com custos reduzidos, de soluções e aplicações que respondam às necessidades do mercado global, bem como à modernização das suas próprias soluções.

O objectivo principal é criar uma base transversal para o desenvolvimento de soluções nas TI sendo essa mesma base, objecto de constantes optimizações e evoluções com especial atenção para as questões de segurança de informação.

É neste processo que surge a necessidade de criação de um Service Broker, um componente que separa a camada de apresentação da camada de lógica de negócio e que pretende convergir num único ponto a responsabilidade de assegurar algum tipo de segurança efectuando controlo de acesso aos serviços. Este serviço genérico constituirá uma camada de segurança para serviços já existentes, e providenciará formas de autenticação, autorização, condições de acesso e auditoria.

Este Service Broker actuará como ponto de entrada para todos os serviços da empresa, adoptando uma arquitectura orientada a serviços (SOA) [9]. Este componente permitirá à empresa ter um maior controlo sobre os serviços que pretende disponibilizar para o exterior e ocultar certas características do mesmo, como é o caso da sua verdadeira localização ou mesmo o verdadeiro tipo de serviço que está a ser disponibilizado (podendo este serviço ser um web service [14] ou algo diferente modificado pelo Service Broker para ser mostrado para o exterior como sendo um web service e se comportar como tal, mas tendo um comportamento interno diferente).

1.3 Apresentação da Organização

1.3.1 História

A AMBISIG – Ambiente e Sistemas de Informação Geográfica, S.A. (adiante designada por AMBISIG), foi criada em 1994 e tem vindo a desenvolver a sua actividade centrada na utilização das Tecnologia de Informação Geográfica, Sistemas de Informação e Consultoria.

No que se refere às Tecnologias de Informação Geográfica, a actividade da AMBISIG cobre as suas múltiplas vertentes, desde o carregamento e estruturação de informação gráfica e alfanumérica, à prestação de serviços de consultoria para

desenvolvimento de projectos locais ou integrados, sendo estes últimos também conhecidos por Sistemas de Informação Geográfica.

Acompanhando a rápida evolução das tecnologias envolvidas, a AMBISIG, desenvolveu e actualmente disponibiliza um conjunto vasto de soluções de acesso e manuseamento de bases de dados geográficas ou tradicionais, em ambientes monoposto ou em sistemas distribuídos via intranet ou internet.

[3]

1.3.2 Actividade

A AMBISIG teve o seu início de actividade em 22 de Julho de 1994 e é uma sociedade anónima com objecto social em projecto de ambiente, paisagismo, arquitectura, engenharia e sistemas de informação geográfica, estudos e consultoria nas mesmas áreas, produção de cartografia, produção de cadastro predial, desenvolvimento de software e soluções e serviços para internet.

A AMBISIG desenvolve soluções para:

- Governo electrónico [eGovernment] (Sistemas de Informação Geográfica, Gestão Documental, Serviços de Atendimento Online multi-canal, ERP, Gestão da Qualidade e Ambiente, Business Intelligence);
- eBank.Solutions para o mercado financeiro.
- MobileMaping para a gestão integrada de infra-estruturas de transporte rodoviário e ferroviário.
- Produtos iCity, que utilizam a plataforma de RDS já existente para a comunicação de dados e interligação dos sistemas.

A AMBISIG posiciona-se no mercado nacional e internacional em duas vertentes distintas:

- Para a Administração Pública: Implementação de soluções de Cadastro, Gestão e Manutenção de infra-estruturas Rodoviárias e Ferroviárias integradas com a digitalização dos seus processos potenciando a visibilidade e o aumento de produtividade;
- Para o Sector Privado: Implementação de soluções integrando a dimensão geográfica com a digitalização dos processos da empresa que permitem o

desenvolvimento de soluções inovadoras que aumentam a criação de valor dessas empresas.

[3]

1.4 Organização do Documento

Este documento está organizado da seguinte forma:

- Capítulo 2 – Especificação dos objectivos do projecto, metodologia de desenvolvimento e plano de trabalhos previsto e realizado.
- Capítulo 3 – Especificação das tecnologias e provas de conceito utilizadas como suporte à realização do projecto
- Capítulo 4 – Apresentação do projecto desenvolvido com especificação das funcionalidades implementadas e testes efectuados ao protótipo obtido.
- Capítulo 5 – Conclusões finais sobre o projecto realizado contendo uma avaliação crítica e apreciações sobre o projecto.
- Bibliografia – Referência a documentação utilizada na elaboração deste documento e outras referências relevantes para uma melhor compreensão do trabalho desenvolvido.
- Anexos – Documentação mais específica e técnica, referida no corpo deste documento, e usada para fornecer uma visão mais detalhada sobre todo o trabalho realizado na obtenção dos objectivos.

Capítulo 2

Objectivos, Metodologia e Planeamento

2.1 Objectivos

De seguida serão expostos todos os objectivos gerais a que este projecto se propunha sendo ainda feita uma breve apresentação de alguns objectivos específicos e motivação por detrás dos mesmos.

2.1.1 Objectivos Gerais

Este projecto tem como principal objectivo a construção de um serviço genérico que constitua uma camada de controlo de acesso para outros serviços já existentes, ou seja, pretende-se construir e convergir num único componente a responsabilidade de providenciar alguma segurança através do condicionamento de acesso a serviços.

Para o efeito, é considerado um ponto intermédio para a disponibilização de serviços: o Service Broker. Este componente será o responsável por responder às solicitações aos serviços disponibilizados, e que, como tal, apenas este conhece. Por outro lado, apenas o Service Broker saberá de que forma estes serviços estarão verdadeiramente disponíveis e onde estes podem ser acedidos. Escondendo desta forma os verdadeiros serviços ao mundo exterior e disponibilizando uma versão diferente destes (mais segura ou alterada de alguma forma).

De uma forma geral o componente a construir deverá satisfazer dois requisitos:

1. Adicionar uma camada de segurança (controlo de acesso) sobre web services [14] já existentes;

2. Publicar um web service com base em *assemblies* [6], adicionando-lhe a mesma camada de segurança identificada em 1.

Ou seja, este componente deverá ser capaz de alterar algumas das características de um web service já existente, para que este seja disponibilizado para o mundo exterior com requisitos de invocação mais restritos que os originais, de forma a permitir algum tipo de controlo de acesso às operações existentes. Para além disso, deverá também ser estudada uma forma deste Service Broker poder expôr *assemblies* (bibliotecas) como web services, sendo gerado um WSDL [11] (onde tal não existe) para estas bibliotecas, permitindo também um controlo de acesso às operações disponíveis. Este WSDL deverá ser idêntico ao usado pelos web services [14], pois trata-se de um documento XML [12] que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis.

A figura 1 representa em alto nível a arquitectura do componente a ser desenvolvido:

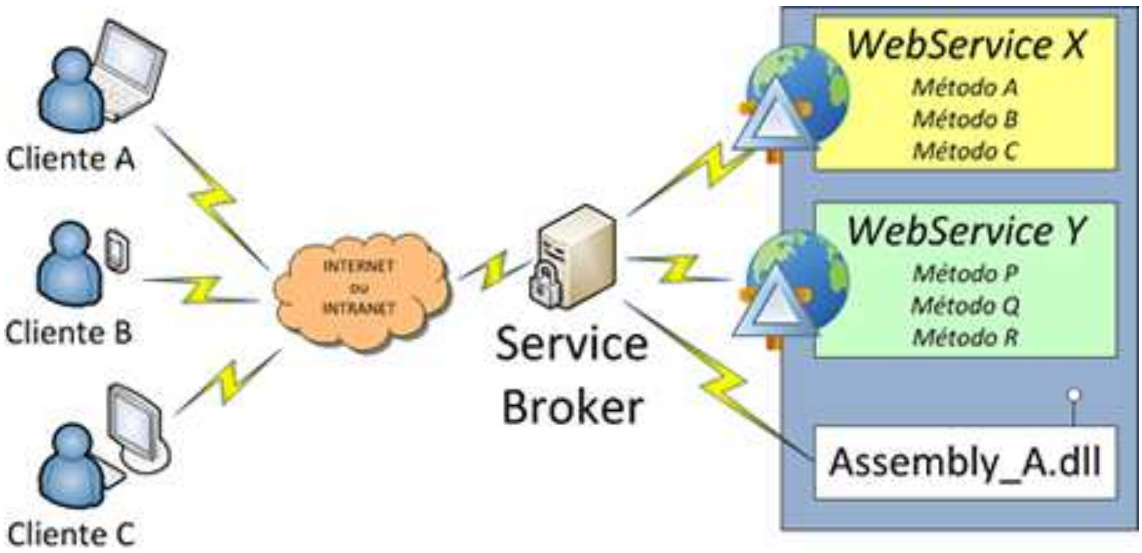


Figura 1: Arquitectura do Service Broker

2.1.2 Objectivos Específicos

O Service Broker desenvolvido irá criar uma camada que estabelece a ligação entre a camada de apresentação (interface com o utilizador) e a camada de lógica de negócio.

Na figura 2 é possível observar que o Service Broker deverá permitir “esconder” a camada de negócio (*Business Layer*), sendo este o responsável por fornecer toda a informação que os clientes necessitam para aceder aos serviços disponibilizados. Neste contexto, consideram-se clientes as aplicações que usam os serviços disponibilizados pelo Service Broker para processarem os pedidos dos seus utilizadores e que apresentam o resultado desse processamento aos autores dos pedidos, estando estes posicionados na camada de apresentação (*Presentation Layer*). Para garantir esse fornecimento de informação, uma forma controlada de WSDL será disponibilizada para informar os clientes do tipo de serviços que estão disponíveis, garantindo que estes conseguem aceder de uma forma controlada a cada serviço, sendo este WSDL modificado ou gerado consoante o serviço a ser acedido internamente.

Com a adição desta nova camada, logo acima da camada de negócio, será possível passar a responsabilidade de controlo de acesso de cada serviço presente na camada de negócio para a camada logo acima, contribuindo para uma implementação mais simples e despreocupada de cada serviço, já que todas as condições de acesso serão controladas pelo Service Broker.

Será também da responsabilidade do Service Broker, e após verificadas as condições de acesso ao serviço, o reencaminho de cada pedido recebido para o respectivo serviço invocado.

A figura 2 mostra onde se posicionará o Service Broker de forma a cumprir a sua função. Este servirá como intermediário entre as aplicações clientes presentes na camada de apresentação e o processamento de informação/pedidos proporcionado pela camada de negócio. Este processamento poderá ou não recorrer a consultas de bases de dados.

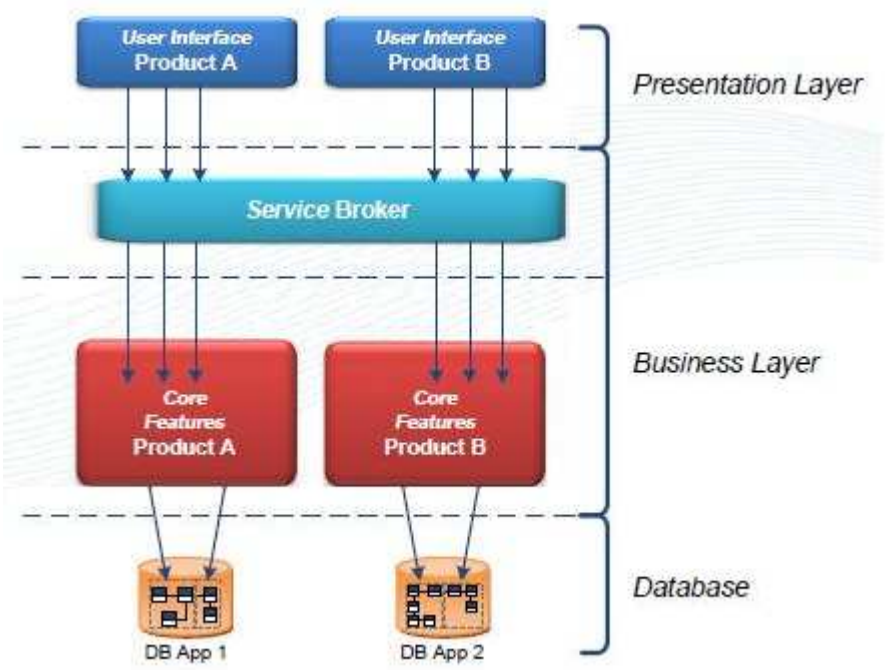


Figura 2: Posicionamento do Service Broker

Assim sendo a construção desta camada irá requisitar a implementação das seguintes operações:

1. Para adicionar a camada de segurança a web services já existentes para efectuar controlo de acesso, e uma vez que já existe um WSDL que identifica o web service, apenas será necessário estender o mesmo de forma a introduzir o requisito de segurança (que não existe no WSDL original);
2. Com base no serviço e nas suas operações deverá ser efectuado um controlo de acesso baseado na premissa de serviço público (acessível) ou serviço privado (acesso restrito);
3. Como o Service Broker será um intermediário entre o cliente e o serviço, deverá ser efectuada uma forma de intercepção e reencaminhamento de mensagens, filtrando as mesmas;
4. Devido à necessidade de se querer publicar *assemblies* como web services, será necessário gerar dinamicamente um WSDL para as operações presentes em cada *assembly*. Para se conhecer que operações disponibiliza uma

assembly, será necessário efectuar a leitura dessa *assembly* recorrendo à técnica de reflexão (*reflection*), que permite fazer uma consulta ao interior de uma *assembly* e inferir o seu conteúdo.

Para garantir um maior controlo sobre os tipos de objectos a serem passados entre os intervenientes, deverá ser estudado uma forma de introduzir *SOAP Extensions* [1] na arquitectura a desenvolver. Esta extensão SOAP (*SOAP Extension*) é, como o próprio nome indica, uma extensão ao formato normal das mensagens SOAP [13], garantindo que uma mensagem SOAP transporta mais informação do que aquela que é definida por defeito pelo protocolo. Este tipo de controlo permitirá ao Service Broker ter um maior conhecimento da estrutura de cada mensagem que circula na rede e que chega até si. Com este conhecimentos mais profundo, será possível tomar decisões tendo em conta não só a estrutura da mensagem mas também o seu conteúdo.

No caso da disponibilização de *assemblies* como web services, deve ser concretizada uma forma de facilmente se proceder à actualização dessa *assembly*. Já que as técnicas normais de carregamento e invocação de *assemblies* provocam o "bloqueio" das mesmas (não é permitido o descarregamento individual de *assemblies* uma vez estando estas carregas no domínio de invocação), deve ser estudada uma forma de se efectuar uma invocação a uma *assembly* sem que esta fique "bloqueada" e para que possa posteriormente ser removida e substituída por outra (efectuando a sua actualização).

Por fim, e de forma a ser possível a rápida configuração e disponibilização dos serviços a que o Service Broker poderá aceder (e que disponibilizará para o exterior), deverá ser desenvolvida uma forma de interface de administração/configuração e de acesso pelos utilizadores. Esta interface, a ser construída sob a forma de portal de acesso, será usada pelo administrador para configurar a informação a ser acedida pelo Service Broker, enquanto um utilizador normal usará esta interface para consultar a informação por este disponibilizada.

2.2 Metodologia de Desenvolvimento

Programação Extrema

Programação Extrema (ou eXtreme Programming, XP) [5] é uma metodologia ágil para equipas pequenas e médias com o objectivo de desenvolver software com requisitos vagos e em constante mudança. Para isso, adopta a estratégia de constante acompanhamento e realização de pequenos ajustes durante o desenvolvimento de software.

Os quatro valores fundamentais da metodologia XP são: comunicação, simplicidade, *feedback* e coragem. A partir desses valores, possui como princípios básicos: *feedback* rápido, presumir simplicidade, mudanças incrementais, abraçar mudanças e trabalho de qualidade.

De entre as variáveis de controlo em projectos, como o custo, tempo, qualidade e escopo (o que se pretende atingir), há um foco explícito em escopo. Para isso recomenda-se a priorização de funcionalidades que representem maior valor possível para o negócio. Desta forma, caso seja necessário a diminuição de escopo, as funcionalidades menos valiosas serão adiadas ou canceladas.

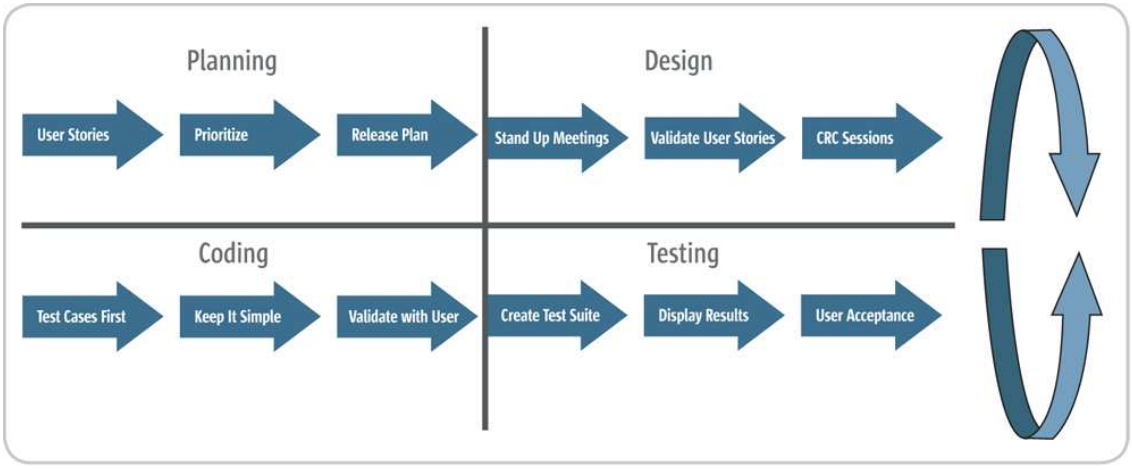


Figura 3: Ciclo de vida da metodologia XP

Para aplicar aos valores e princípios durante o desenvolvimento de software, XP propõe uma série de práticas, havendo uma confiança muito grande na sinergia entre elas, onde os pontos fracos de cada uma são superados pelos pontos fortes de outras.

Planning Game: O desenvolvimento é feito em iterações semanais, onde o cliente identifica prioridades e os desenvolvedores as estimam.

Small Releases: A liberação de pequenas versões funcionais do projecto possibilita a execução de testes a parte do sistema por parte do cliente.

Metaphor: Procura facilitar a comunicação entre o cliente e o programador, efectuando uma tradução das palavras do cliente para o significado que este espera dentro do projecto.

Simple Design: Simplicidade é um princípio da XP, não confundindo código simples com código fácil. Nem sempre o código mais fácil levará à solução mais simples por parte do projecto. Esse entendimento é fundamental para o bom funcionamento da XP para que código fácil deva ser substituído por código simples.

Whole Team: A equipa de desenvolvimento é formada pelo cliente e pela equipa de desenvolvimento.

Custormer Tests: Testes construídos pelo cliente, conjunto de analistas e testadores, para aceitar um determinado requisito do sistema.

Sustainable Pace: Trabalhar com qualidade procurando um ritmo de trabalho saudável.

Stand-up Meeting: Reuniões em pé para não se perder o foco nos assuntos, produzindo reuniões rápidas, apenas abordando tarefas realizadas e tarefas a realizar pela equipa.

Coding Standards: Estabelecimento de regras de programação de forma a gerar um código fonte aparentemente editado pela mesma pessoa, mesmo quando a equipa possui 10 ou 100 membros.

Test Driven Development: Criação de testes unitários antes de criar o código para que os testes sejam executados.

Refactoring: Processo de melhoria continua da programação, com o mínimo de introdução de erros e mantendo a compatibilidade com o código já existente

Continuous Integration: Sempre que uma nova funcionalidade é produzida, integra-la de imediato na versão actual do sistema, reduzindo assim a possibilidade de

conflitos e erros no código fonte. Integrar de forma contínua permite saber o status real da programação.

2.3 Planeamento

Nesta secção serão expostos dois planeamentos: o planeamento inicialmente apresentado como sendo "o mapa" a usar para a elaboração do projecto, e o planeamento efectivamente concretizado, sendo este último uma versão mais detalhada de todas as actividades desenvolvidas.

Será ainda efectuada uma comparação entre ambos os planeamentos, constatando a ocorrência de desvios ao plano inicialmente proposto e as razões para essas ocorrências.

2.3.1 Planeamento Inicial

A calendarização de tarefas inicialmente prevista para alcançar os objectivos enunciados é apresentada na figura 4.

	Task Name	Duration	Start	Finish
1	**Implementação de um Service Broker**	**180,5 days?**	**Thu 10-09-09**	**Thu 20-05-10**
2	**Fase de Iniciação**	**18,5 days?**	**Thu 10-09-09**	**Tue 06-10-09**
3	Análise dos requisitos a serem implementados	30 hrs?	Thu 10-09-09	Tue 15-09-09
4	Contextualização com o ambito da actividade	30 hrs?	Tue 15-09-09	Mon 21-09-09
5	Realização de provas de conceito iniciais	70 hrs?	Tue 22-09-09	Fri 02-10-09
6	Escrita do Relatório Preliminar	14 hrs?	Fri 02-10-09	Tue 06-10-09
7	**Fase de Execução**	**135 days?**	**Tue 06-10-09**	**Tue 13-04-10**
8	Implementação da versão base do Service Broker	54 hrs?	Tue 06-10-09	Thu 15-10-09
9	Implementação de cliente WSDL sem utilização de proxy	216 hrs?	Thu 15-10-09	Mon 23-11-09
10	Utilização de SOAP Extension e SOAP Fault	108 hrs?	Mon 23-11-09	Thu 10-12-09
11	Execução de logs de utilização do Serviço	216 hrs	Thu 10-12-09	Mon 18-01-10
12	Criação dinâmica de WSDL para componentes sem WebService	216 hrs?	Mon 18-01-10	Wed 24-02-10
13	Integração com WCF	270 hrs?	Wed 24-02-10	Tue 13-04-10
14	**Fase de Conclusão**	**27 days?**	**Tue 13-04-10**	**Thu 20-05-10**
15	Benchmarking ao Service Broker	64,8 hrs?	Tue 13-04-10	Fri 23-04-10
16	Execução de testes unitários	86,4 hrs?	Fri 23-04-10	Mon 10-05-10
17	Elaboração do Relatório Final	64,8 hrs?	Mon 10-05-10	Thu 20-05-10

Figura 4: Planeamento de tarefas inicial

Nesta calendarização apenas estavam enunciadas as principais metas a atingir não sendo especificadas nenhumas das suas subtarefas nem o tempo estimado para a sua realização.

2.3.2 Planeamento Revisto

Nesta calendarização revista, representada pela figura 5, é apresentada uma especificação das subtarefas realizadas, e o seu tempo de execução, de forma a atingir as principais metas identificadas na calendarização inicial.

Neste planeamento é ainda enunciado as alterações ocorridas ao planeamento inicial, como no caso da introdução de mais *milestones* na fase de execução, o reordenamento de algumas tarefas e a eliminação da etapa responsável por fazer a integração com o WCF [8], um modelo de programação unificado e ambiente de execução criado pela Microsoft com o objectivo de criar aplicações orientadas a serviços que comunicam na web. Esta tinha como principal objectivo a exploração de uma nova tecnologia da Microsoft, o WCF, com o intuito de estender o tipo de comunicação entre os clientes do Service Broker e o próprio Service Broker para além do habitual HTTP+XML providenciado pelo SOAP, estendendo essa comunicação ao uso de TCP, UDP, filas de espera (message queuing), entre outros [8].

O Mapa de Gantt associado a este planeamento revisto é disponibilizado no Anexo 1 deste documento.

		Task Name	Duration	Start	Finish
1		**Implementação de um Service Broker**	**196 days**	**Thu 10-09-09**	**Thu 10-06-10**
2		**Fase de Iniciação**	**40 days**	**Thu 10-09-09**	**Wed 04-11-09**
3	✓	Análise dos requisitos a serem implementados	5 days	Thu 10-09-09	Wed 16-09-09
4	✓	Contextualização com o ambito da actividade	3 days	Thu 17-09-09	Mon 21-09-09
5	✓	**Realização de provas de conceito iniciais**	**32 days**	**Tue 22-09-09**	**Wed 04-11-09**
6	✓	Cliente de WSDL	7 days	Tue 22-09-09	Wed 30-09-09
7	✓	HTTPHandler para captura de tráfego	7 days	Thu 01-10-09	Fri 09-10-09
8	✓	Uso de SOAP Extensions	4 days	Mon 12-10-09	Thu 15-10-09
9	✓	Construção Dinâmica de WSDL	9 days	Fri 16-10-09	Wed 28-10-09
10	✓	Uso do AppDomain	5 days	Thu 29-10-09	Wed 04-11-09
11	✓	Escrita do Relatório Preliminar	5 days	Mon 09-11-09	Fri 13-11-09
12		**Fase de Execução**	**125 days**	**Thu 05-11-09**	**Wed 28-04-10**
13	✓	Estudo dos casos de uso	3 days	Thu 05-11-09	Mon 09-11-09
14	✓	Elaboração de Diagramas de Sequência	7 days	Tue 10-11-09	Wed 18-11-09
15	✓	Elaboração de Diagramas de Fluxo	8 days	Thu 19-11-09	Mon 30-11-09
16	✓	Implementação da versão base do Service Broker	15 days	Tue 01-12-09	Mon 21-12-09
17	✓	Utilização de SOAP Extension e SOAP Fault	4 days	Tue 22-12-09	Fri 25-12-09
18		**Criação dinâmica de WSDL para componentes sem WebService**	**10 days**	**Mon 28-12-09**	**Fri 08-01-10**
19	✓	Forma de usar appDomain	5 days	Mon 28-12-09	Fri 01-01-10
20		Geração de WSDL em appDomain	5 days	Mon 04-01-10	Fri 08-01-10
21	✓	Criação de WebServices de administração	10 days	Mon 11-01-10	Fri 22-01-10
22		**Criação de um portal de administração**	**30 days**	**Mon 25-01-10**	**Fri 05-03-10**
23		Criação de um serviço	10 days	Mon 25-01-10	Fri 05-02-10
24	✓	Listagem de Serviços	5 days	Mon 08-02-10	Fri 12-02-10
25	✓	Visualização dos dados de um Serviço	5 days	Mon 15-02-10	Fri 19-02-10
26	✓	Listagem de Métodos	5 days	Mon 22-02-10	Fri 26-02-10
27		Formulário de teste a um Método de um Serviço	5 days	Mon 01-03-10	Fri 05-03-10
28	✓	Execução de logs de utilização do Serviço	5 days	Mon 08-03-10	Fri 12-03-10
29	✓	Implementação de permissões nos métodos disponibilizados	7 days	Mon 15-03-10	Tue 23-03-10
30	✓	Implementação de níveis de segurança nos serviços disponibilizados	7 days	Wed 24-03-10	Thu 01-04-10
31	✓	Adaptação do portal de administração para uso generalizado	10 days	Fri 02-04-10	Thu 15-04-10
32		Invocação de componentes sem WebService	6 days	Fri 16-04-10	Fri 23-04-10
33		Geração dinâmica de WSDL de acordo com permissões do utilizador	3 days	Mon 26-04-10	Wed 28-04-10
34	✓	**Fase de Conclusão**	**8 days**	**Thu 29-04-10**	**Mon 10-05-10**
35	✓	Benchmarking ao Service Broker	4 days	Thu 29-04-10	Tue 04-05-10
36	✓	Execução de testes unitários	4 days	Wed 05-05-10	Mon 10-05-10
37	✓	Elaboração do Relatório Final	23 days	Tue 11-05-10	Thu 10-06-10

Figura 5: Planeamento de tarefas revisto

Durante a execução de um projecto podem surgir situações que levam a uma reestruturação total ou apenas a algumas alterações do planeamento inicialmente previsto de forma a atingir os objectivos identificados. Este projecto não foi excepção. Embora não tenha sido necessário algo tão radical como uma reformulação total do planeamento inicial, foi necessário proceder a alguns acertos ao mesmo de forma a cumprir prazos estabelecidos e para que fosse dedicado mais tempo às funcionalidades identificadas como mais importantes e indispensáveis ao projecto em construção.

A inserção de mais actividades e a remoção de outras na fase de execução é um bom exemplo de acertos efectuados a este projecto. Este deveu-se à decisão de não tornar a integração com WCF como prioritária e dedicar mais tempo à criação de um

portal de administração, bem como a configuração dos serviços acedidos pelos utilizadores e suas permissões.

A integração com WCF já se encontrava definida à partida como sendo a tarefa a sacrificar, pois era sabido que esta iria necessitar de provas de conceito adicionais para sua correcta implementação, o que provocaria novas iterações sobre as actividades já desenvolvidas, ultrapassando os nove meses definidos como tempo limite para a realização do estágio. Esta tarefa constava do planeamento inicial para dar uma ideia mais abrangente do tipo de caminho a seguir após a realização das principais tarefas.

A criação de um portal de administração foi outra das grandes alterações introduzidas ao plano inicial. No decorrer do projecto tornou-se claro que deveria ser fornecida alguma forma de configuração do componente principal em desenvolvimento, e que esta deveria ser o mais simples e intuitiva possível. Esta forma de configuração consistiu na criação de um portal de administração que mais tarde se viria a tornar num portal de acesso já que este permitia tanto o acesso por parte de administradores como de utilizadores do Service Broker.

No caso da fase de inicialização, esta foi executada em mais tempo do que o previsto devido à sua componente de investigação e para que a fase de execução fosse realizada sobre uma base de conceitos sólidos.

Capítulo 3

Ferramentas, SOAP e Provas de Conceito

Neste capítulo serão apresentadas as ferramentas, o protocolo de acesso a objectos e as provas de conceito que serviram como base ao desenvolvimento do Service Broker.

3.1 Ferramentas

Ao longo da elaboração deste projecto foram usadas várias ferramentas, tanto para o desenvolvimento de *software*, como para a especificação de algumas tarefas que iam sendo realizadas. De seguida é apresentada uma listagem com as ferramentas usadas, breve descrição sobre as mesmas e contexto em que estas foram utilizadas:

- **Microsof Visual Studio 2008** – Usado como principal plataforma de desenvolvimento de *software* [4].

  Foi usada a variante Team Foundation Server SP1 por esta usar e permitir a colaboração entre vários elementos de uma equipa e por permitir um melhor controlo sobre todas as versões previamente lançadas de um determinado produto.

  Embora não tenha sido utilizada a componente de colaboração entre vários elementos de uma equipa, uma vez que todo o desenvolvimento ficou a cargo de apenas uma pessoa, a componente de controlo de versões revelou-se bastante útil uma vez que permitia retroceder para uma versão mais antiga do que aquela em que se estava a trabalhar. Esta possibilidade revelou-se bastante eficaz quando uma nova funcionalidade era introduzida e esta entrava em conflito com as já desenvolvidas.

- **Microsoft SQL Server 2005 Express** – Usado para criar um repositório de dados a ser usado pela aplicação desenvolvida [15].

  Através desta ferramenta foi possível a criação, edição e alteração de uma base de dados, indispensável para o funcionamento da aplicação. Com esta ferramenta foi ainda possível elaborar alguns sripts e testar comandos SQL antes de estes serem utilizados na aplicação.

- **Internet Information Services** – Servidor Web criado pela Microsoft para a geração de páginas HTML dinâmicas usando a tecnologia proprietária Active Server Pages (ASP) [2]

  Este permite também gerir o ASP.NET, formato usado por dois tipos de aplicações:

  Paginas Web – tradicionalmente acedidas por utilizadores e com a extensão ASPX

  Web Services – funções disponibilizadas como serviços possíveis de invocar e acessíveis em páginas com extensão ASMX

- **Microsoft Office –** Usado para elaborar a documentação necessária ao projecto.

  Word – Criação de guiões de utilizador, relatórios e especificação de funcionalidades.

  Excel – Introdução dos tempos nos testes efectuados e geração dos gráficos de resultados.

  Visio – Criação de caso de uso, diagramas de sequência e diagramas de fluxo.

  Outlook – Usado como principal meio de comunicação.

3.2 Protocolo de Acesso a Objectos (SOAP)

Uma vez que o SOAP irá ser aquele que permitirá efectuar toda a comunicação, não só entre serviços mas também com os clientes, interessa aqui expôr a sua forma de funcionamento para que mais tarde se torne claro possíveis extensões ou manipulações ao mesmo.

O *Simple Object Access Protocol* (SOAP) é o protocolo de comunicação padrão usado na troca de mensagens quando se necessita de independência da plataforma e da linguagem de programação (web services) [13]. As mensagens SOAP são especificadas em XML e enviadas sobre HTTP. A figura 6 [1] ilustra as etapas por que passa uma mensagem SOAP para efectuar a invocação de um serviço e onde esta mensagem pode ser manipulada para incluir mais informação do que aquela que é introduzida por defeito pelo protocolo de acesso a objectos (SOAP).

Sempre que um cliente (aplicação que usam os serviços) pretende comunicar com um servidor (máquinas responsáveis por disponibilizar serviços e processar pedidos) este pede à sua proxy para decompor um objecto no seu formato XML e prepará-lo para ser enviado pela rede (mensagem SOAP), sendo este processo conhecido como serialização de objectos (fase 1 da figura 6). Após esta serialização é possível modificar a mensagem gerada, podendo ser adicionada ou retirada informação à mesma com o formato XML (SOAP Extension), garantindo sempre que essa modificação poderá ser interpretada pelo servidor que irá processar a mensagem. Uma vez gerada a mensagem SOAP, esta é enviada pela rede para o servidor responsável por processar o pedido. Recebida a mensagem, esta deve ser analisada de forma a verificar a modificação ou não da mensagem SOAP padrão. Feita essa verificação é altura de converter essa mensagem e conteúdo XML no seu objecto ou objectos equivalentes para que possam ser processados pelo servidor. A este processo dá-se o nome de desserialização de objectos (fase 2 da figura 6). Estando o pedido processado e tendo sido obtida uma resposta, dá-se o processo inverso do pedido agora para a resposta. Esta é serializada e preparada para ser enviada ao cliente (fase 3 da figura 6), sendo esta mensagem SOAP manipulada (ou não) após esta serialização, um pouco à semelhança do que aconteceu na fase 1. A mensagem é enviada pela rede para o autor do pedido inicial. Esta é recebida, verificada, manipulada e/ou preparada para a conversão de XML para objecto(s), dando-se a desserialização (fase 4 da figura 6) da mensagem. Por fim essa mensagem/resposta obtida é passada para o cliente para que este a possa processar ou apresentar.

Na figura 6 pode ser consultada a forma de funcionamento acima descrita.

---

[1] Obtida a partir do site MSDN para ilustrar o ciclo de vida de uma mensagem SOAP e onde esta pode ser manipulada de forma a incluir mais informação que a mensagem padrão (SOAP Extension).

MSDN: Visual Studio Developer Center - http://msdn.microsoft.com/en-us/vstudio/default.aspx.

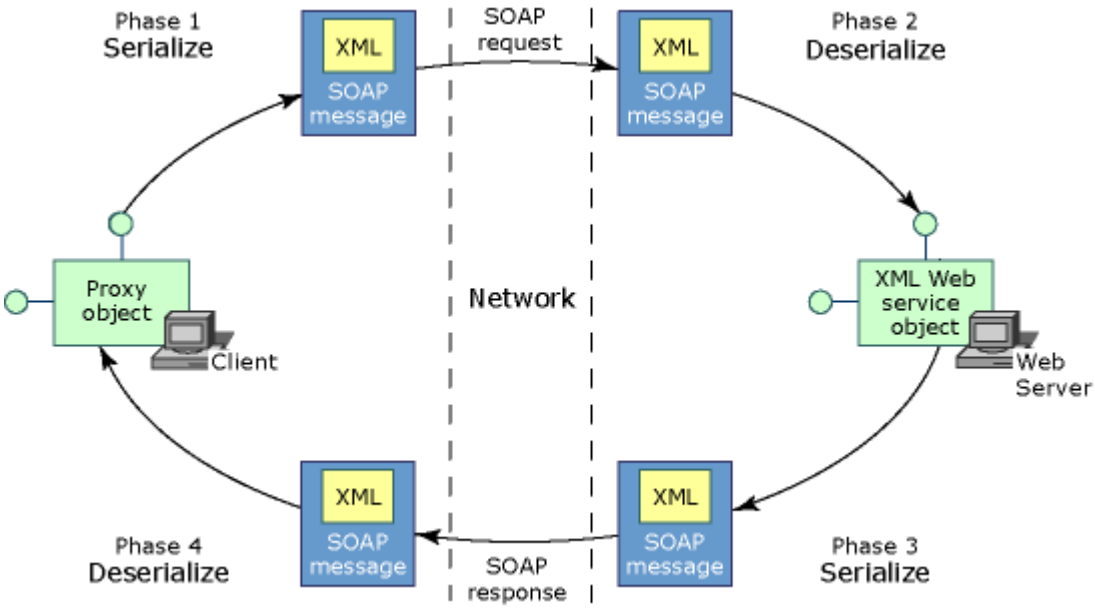


Figura 6: Modo de funcionamento SOAP

Após a compreensão da forma de funcionamento deste protocolo, foi possível passar à elaboração de provas de conceito concretas, tendo sempre como base os passos anteriormente descritos nos casos em que era necessário o processamento de mensagens.

3.3 Provas de Conceito

No que diz respeito à análise efectuada antes da execução efectiva, esta consistiu maioritariamente em testar alguns conceitos de forma a verificar se estes eram concretizáveis e, caso o fossem, como se poderiam implementar. Assim, e de acordo com o plano de trabalhos, o trabalho inicial foi focado na execução de provas de conceito de forma a fundamentar as decisões de implementação. Provas essas iniciadas após reuniões de trabalho de forma a especificar metas a atingir com cada uma das provas de conceito.

3.3.1 1ª Prova: Cliente de WSDL

Esta primeira prova consistia na criação de um cliente que, através de um WSDL disponibilizado, fosse capaz de invocar os métodos que eram definidos pelo serviço como possíveis de invocar.

De forma a alcançar este primeiro objectivo foi criado um web service com 3 métodos possíveis de invocar. Dois desses métodos eram similares, um retornava uma String e o outro retornava um Inteiro, e um último que recebia um inteiro como argumento e retornava um outro inteiro.

Uma vez elaborado este serviço, passou-se à fase da criação de um cliente que apenas conhecendo o WSDL deste web service, fosse capaz de invocar os métodos do mesmo. Para o efeito foi criado um cliente com a interface da figura 7.

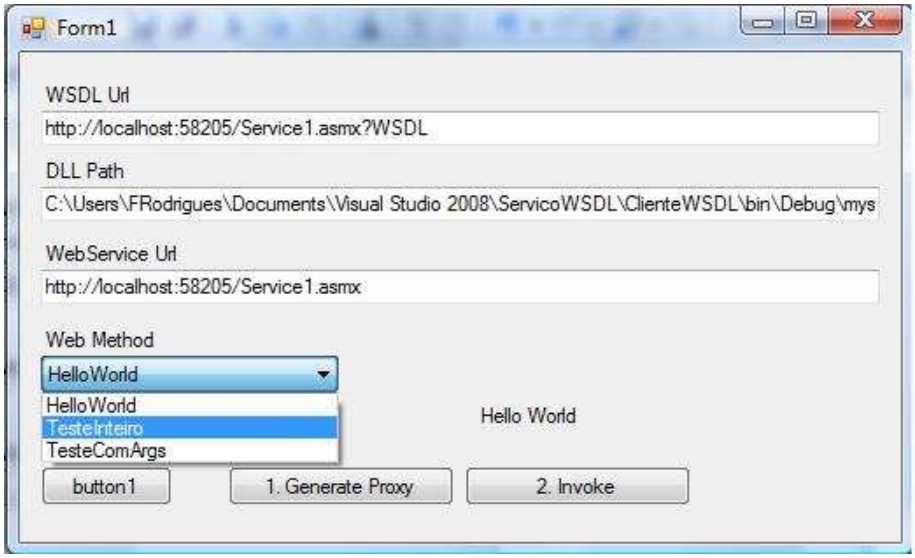


Figura 7: Interface Cliente WSDL

O objectivo deste cliente era que, apenas mediante da disponibilização do caminho para o WSDL do serviço, este fosse capaz de gerar a sua proxy de forma a comunicar com o serviço em questão. Assim, mediante o correcto preenchimento do campo "WSDL Url", era possível premir o botão "1.*Generate Proxy*" que era responsável por gerar a proxy para o web service. Uma vez gerada a proxy, e uma vez estando esta disponível para uso, era possível escolher dos métodos disponíveis, qual aquele que deveria ser invocado, sendo para isso premido o botão "2.*Invoke*". Este invocava o

método e apresentava o seu resultado na *Label* imediatamente acima do respectivo botão de invocação.

De seguida serão explicados em pormenor todos os passos envolvidos na criação da proxy e na invocação do serviço:

1) *Generate Proxy* – Para criar a proxy foi usada a abordagem: wsdl-> cs-> dll, ou seja, mediante um ficheiro WSDL, foi gerado o seu código correspondente em C# recorrendo à ferramenta wsdl.exe (gerador de código através de ficheiros de contracto WSDL, XSD Schema e documentos de descoberta *discomap*) usando o comando:

```
wsdl.exe  /:out:MyService.cs http://localhost:58205/Service1.asmx?WSDL.
```

Este comando gerava o ficheiro *MyService.cs* que por sua vez seria usado para criar a biblioteca *dll* desse mesmo ficheiro, recorrendo à ferramenta csc.exe (compilador de ficheiros C#) usando para isso o comando:

```
csc.exe /target:library /out:MyService.dll MyService.cs.
```

Este comando gerava o ficheiro *MyService.dll* que seria então a proxy responsável pela invocação dos métodos do web service.

Por fim eram verificados quais os métodos possíveis de serem invocados e estes eram disponibilizados na *Dropbox* em "Web Method".

2) *Invoke* – De forma a ser possível a invocação dos métodos, foi necessário usar as técnicas de reflexão (*reflection*) e "Assembly" disponibilizadas pela biblioteca .NET. Estas técnicas permitem em tempo de execução saber, por exemplo, quais os métodos possíveis de serem invocados num ficheiro previamente compilado e onde não se sabe à partida o código fonte que gerou esse mesmo ficheiro.

Uma vez previamente gerado o *MyService .dll*, foi possível criar um objecto por reflexão que instancia-se o serviço remoto o que por sua vez permitiu invocar os métodos sobre esse mesmo objecto.

Este exercício permitiu verificar o funcionamento de alguns sub-conceitos. Como é o caso da geração e compilação em tempo de execução de código C#, que por sua vez seria usado, também em tempo de execução, recorrendo às técnicas de "Assembly" e

reflexão. Sendo assim possível a criação de uma proxy que pudesse conhecer e invocar serviços apenas com o conhecimento de um WSDL.

3.3.2 2ª Prova: .ashx para captura de tráfego

O principal objectivo nesta prova de conceito era o de perceber a teoria por detrás dos *HTTP Handlers*: forma de funcionamento, capacidades e forma de programação/manipulação de dados.

A teoria associada a esta prova de conceito era a seguinte: poderia existir um "Master Server" que era responsável por saber que serviços tinha disponíveis e que métodos um cliente poderia invocar. O cliente por sua vez só tinha conhecimento de que determinados métodos podiam ser invocados, sendo da responsabilidade do "Master Server" de reencaminhar esses pedidos para cada um dos serviços (sendo ainda possível ser da responsabilidade deste, alguma forma de segurança, garantindo que determinado cliente só poderia aceder a um método se este tivesse permissões para tal).

A ideia geral anteriormente descrita está presente na figura 8.

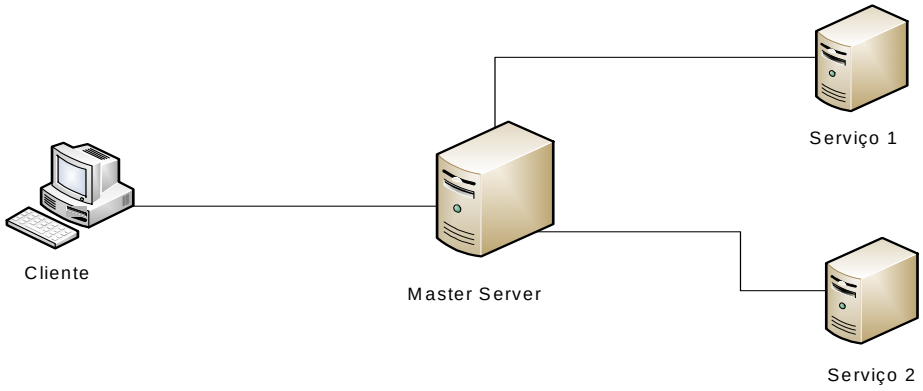


Figura 8: Arquitectura Global

Numa primeira fase, e de maneira a testar a forma como o "Master Server" iria interceptar, tratar e reencaminhar o pedido para o respectivo serviço, apenas foram disponibilizados os métodos possíveis de serem invocados de um serviço, mas passando pelo *HTTP handler* do "Master Server". Assim de forma a testar todo o conceito foi utilizado o cliente da 1ª prova de conceito, sendo apenas alterado o URL onde este iria

obter o serviço possível de ser invocado. Este URL passou a ser algo do género http://localhost:58205/MyWSHandler.ashx (o URL passou a ser a página .ashx em vez da pagina .asmx, de forma a que o pedido pudesse passar pelo *HTTP handler* em vez de ir directamente para onde o serviço se encontrava publicado). Desta forma o cliente manteve-se inalterado (excepção feita para o URL para onde o cliente "aponta") sendo todas as alterações feitas ao nível do *HTTP handler* no "Master Server".

A forma como este *HTTP handler* foi implementado consistiu no seguinte: sempre que o *HTTP handler* recebe uma mensagem SOAP, este retira o corpo dessa mesma mensagem, onde se encontra qual o método a ser invocado e respectivos argumentos (se existirem), e passa o "Master Server" a comportar-se como um cliente para cada um dos serviços que disponibiliza, fazendo a invocação do método recebido e aguardando a sua resposta.

Uma vez recebida a resposta por parte do método invocado, o "Master Server" deve agora reencaminhar esta para o cliente, autor da invocação. Deve para isso criar uma mensagem SOAP que o cliente perceba e possa interpretar, finalizando assim o processo de invocação de serviços recorrendo a um *HTTP handler* (intercepção de mensagens).

De forma a melhor representar como todo o processo foi implementado, este é mostrado de um forma gráfica na figura 9.

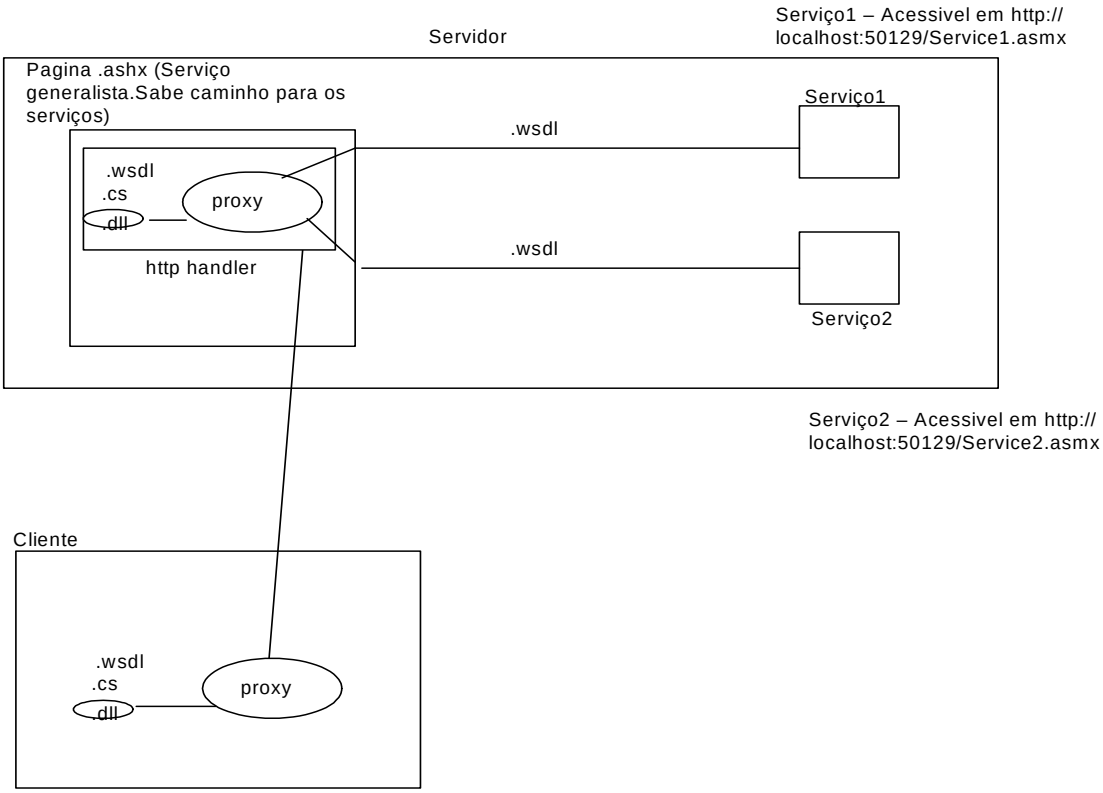


Figura 9: HTTP Handler para captura de tráfego

Atenções finais para a forma como as proxies são criadas: esta ainda se baseia na forma wsdl->cs->dll. processo esse descrito na 1ª prova de conceito. Ainda referir que agora este processo ocorre em 2 lados, no lado do cliente (como já acontecia) e do lado do "Master Server" (ao nível do *HTTP handler*), pois este terá de se comportar como cliente e como servidor, servidor dos clientes que querem aceder aos serviços e cliente dos serviços que disponibiliza, garantindo assim a satisfação aos pedidos dos "clientes externos".

3.3.3 3ª Prova: SOAP Extensions

Nesta terceira prova de conceito o objectivo era criar uma forma de autenticação por parte do utilizador perante o web service, verificando assim se este tinha autorização para a sua invocação.

Para esse efeito foi utilizado uma forma simples do SOAP Extensions [1]. Mais precisamente foi utilizado o SOAP Header [13], uma forma disponibilizada pelo .NET Framework 3.0 de facilmente se implementar esta forma de SOAP Extension.

A forma implementada consistiu no seguinte: um serviço sempre que requer alguma forma de autenticação, dá-o a conhecer através do WSDL que disponibiliza. Na proxy que cada cliente gera para aceder aos serviços existe um meio de este criar uma mensagem específica (incluir o SOAP Header na mensagem, e este levar toda a informação necessária para a autenticação de cada cliente a um serviço) para um determinado serviço.

Uma vez criada essa mensagem, esta é enviada para o "Master Server" que a desencapsula e utiliza uma das suas proxies para aceder ao serviço requisitado (actuar como cliente dos serviços que disponibiliza). Uma vez na posse de todos os elementos, o "Master Server" encontra-se em condições de invocar o serviço, que verificará as credenciais fornecidas. Se o utilizador tiver permissões para aceder ao serviço este responderá ao pedido, caso contrário o serviço responderá com uma excepção (sem permissões).

Aqui fica o exemplo de uma mensagem com SOAP Header:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/
                                  envelope/">
  <soap:Header>
    <AuthHeader xmlns="http://tempuri.org/">
      <UserName>teste</UserName>
      <Password>teste1</Password>
    </AuthHeader>
  </soap:Header>
  <soap:Body>
       <HelloWorld xmlns="http://ambisig.pt" />
  </soap:Body>
</soap:Envelope>
```

Nesta primeira fase exploratória existem ainda algumas formas de optimização, como é o caso da verificação de credenciais se processar ao nível do "Master Server" e não do serviço. Se esta verificação acontecer no "Master Server", a invocação que este faz ao serviço poderá ser diferente da invocação do cliente ao "Master Server" (não necessitar de incluir o SOAP Header, uma vez que a verificação já tinha sido efectuada). Por outro lado esta dupla verificação pode acrescentar mais um nível de segurança, na medida em que um utilizador não autorizado possa comprometer o Master Server e

passar a sua verificação, mas não conseguir aceder mesmo assim ao serviço, pois este tem também uma verificação a fazer.

Foram ainda efectuados alguns testes a uma forma mais completa e abrangente de implementar SOAP Extensions: a criação de uma classe que deriva da classe SoapExtension. No entanto devido a alguma falta de documentação a explicar a sua correcta implementação impossibilitou um exemplo útil de forma de utilização.

Numa segunda fase, e após alguns esclarecimentos providenciados pelo coordenador do projecto, foi então criado o WSDL de forma a criar a desejada camada de segurança, agora ao nível do “Master Server”. Sendo este o responsável por verificar as credenciais de acesso aos serviços.

Depois de se compreender como era criado o WSDL com requisito de SOAP Header, passou-se à criação desse mesmo WSDL, mas agora do lado do “Master Server” (até aqui todos os WSDL tinham sido gerados do lado dos serviços), de forma a que fosse este a requisitar a autenticação do utilizador e não o serviço em si (o utilizador apenas se autentica no nó intermédio que usa para aceder ao serviço, sem se aperceber que existe esse nó intermédio). Uma vez disponibilizado este WSDL ao cliente, este gera uma proxy para usar o serviço (com requisito de SOAP Header).

Depois de invocado o serviço com as respectivas credenciais, o “Master Server” desencapsula a mensagem, verifica as credenciais, e se o utilizador tiver permissões para aceder ao serviço, o pedido é reencaminhado para o respectivo serviço, caso contrário o serviço nem sequer é invocado e o utilizador é informado de que não possui permissões para aceder ao serviço.

Um exemplo de como tudo se processa é fornecido pelo esquema da figura 10.

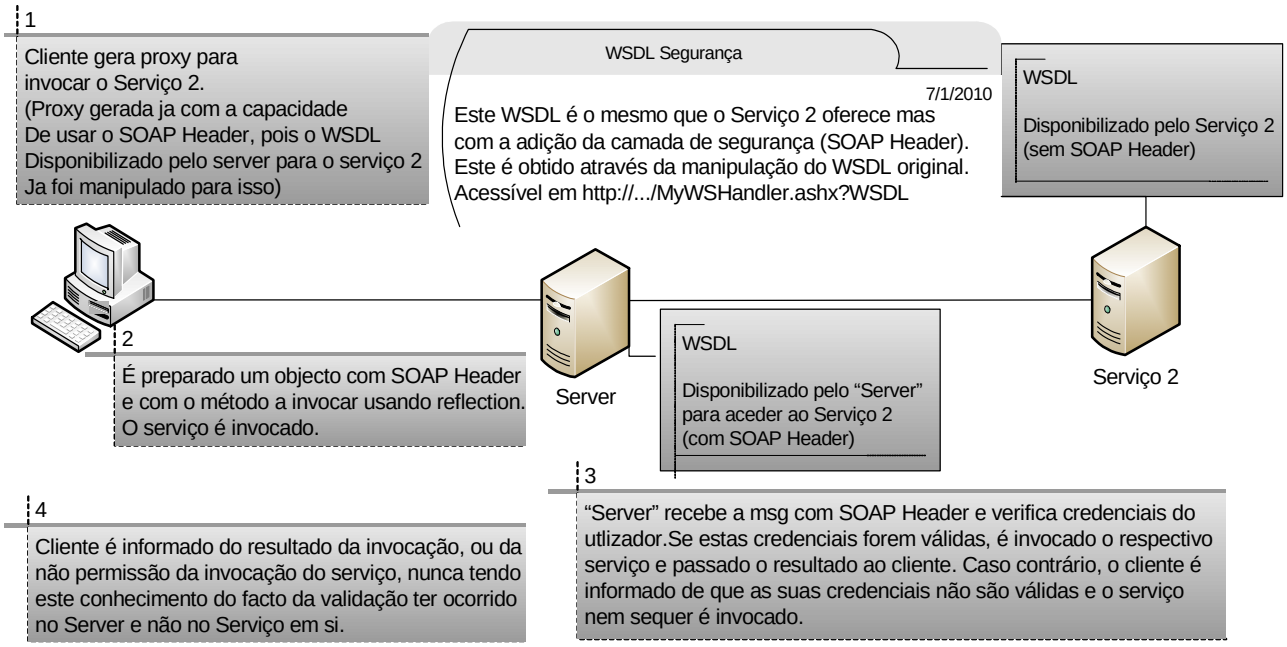


Figura 10: Camada de segurança ao nível do "Master Server"

Como nota final apenas referir o facto de os dois serviços disponibilizados para testes, terem de ser identificados na altura da invocação através da distinção dos mesmos no URL sendo que os métodos do serviço 1 estavam acessíveis em `http://.../MyWSHandler.ashx?SID=1` e o método do serviço 2 estava acessível através do URL `http://.../MyWSHandler.ashx?SID=2`.

3.3.4 4ª Prova: Construção dinâmica de WSDL usando reflexão

Esta prova de conceito tinha como por base uma ideia muito simples: dado um ficheiro qualquer compilado previamente numa biblioteca *dll*, ler esse ficheiro usando a técnica de reflexão (*reflection*), de forma a saber que métodos estavam presentes nesse ficheiro de forma a serem usados na construção do WSDL. Essa construção deveria ser o mais dinâmica possível para que, mudando apenas o ficheiro *dll* fosse sempre possível reproduzir um novo WSDL válido (que pudesse ser usado pelos clientes para gerar a sua proxy de forma a invocar o serviço).

A figura 11 ilustra esta mesma ideia de uma forma gráfica:

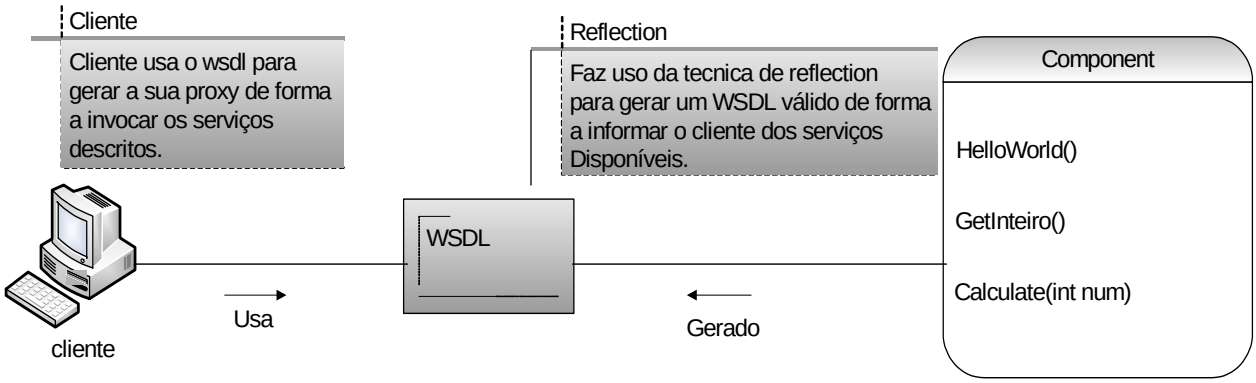


Figura 11: Criação dinâmica de WSDL

Todo o trabalho, neste caso, foi concentrado na geração do WSDL, sendo usado como cliente a aplicação previamente desenvolvida para as provas de conceito anteriores, apenas com ligeiras modificações.

Assim todo o trabalho começou por criar alguns web services simples, que numa primeira fase apenas recebiam como argumento objectos primários, o mesmo acontecendo com o seu retorno. De seguida procedeu-se à análise do WSDL gerado automaticamente pela ferramenta Visual Studio de forma a perceber como este documento descrevia um web service, para que a proxy pudesse ser gerada e esta pudesse comunicar sem problemas.

Feita esta análise, passou-se à tentativa de gerar este mesmo documento por código. Inicialmente apenas foi gerada uma cópia exacta do WSDL disponibilizado, de forma a ter uma estrutura sólida para acrescentar/mudar mais tarde, recorrendo á técnica de reflexão.

Uma vez produzido este WSDL “hard coded”, e tendo-se acertado com os vários nós no sitio certo, procedeu-se então à tentativa de integração da técnica de reflexão neste mesmo pedaço de código.

Para o efeito foi criada uma classe com apenas um método, o método *HelloWorld*. De seguida esta classe foi compilada, sendo gerado um ficheiro com extensão *dll*. Uma vez gerado esse ficheiro, assumiu-se que não se sabia que métodos este continha para

que fosse usada a técnica de reflexão (*reflection*) neste simples ficheiro para se tentar perceber de que forma se poderia usar esta técnica para saber quais os métodos que se encontravam no seu interior. Após se perceber a forma de funcionamento desta técnica e estando na posse dos nomes dos métodos, era então possível aceder a toda a informação dos mesmos (se tinham argumentos, quais eram esses argumentos, de que tipo e qual o tipo de retorno do método).

Estando-se na posse da informação de cada método, apenas foi necessário estruturar o código de forma a acolher esta informação tantas vezes quantos os métodos existentes, colocando sempre essa informação no sítio certo.

No final foi produzido um pedaço de código que, mediante o namespace onde se pretende trabalhar, o ficheiro *dll* e o nome do serviço, gera um WSDL válido que pode ser usado por um cliente para gerar a sua proxy e esta ser usada para comunicar com os serviços.

Os principais passos tomados na criação de um pedaço de código que permite gerar um WSDL são apresentados de seguida, sendo o WSDL que serviu como "inspiração" para a criação desse código disponibilizado no Anexo 2 deste documento.

Começa-se por criar um documento XML e preenchê-lo com os namespaces que irão ser necessários/usados no documento. De seguida é usada a técnica de reflexão para saber quais os métodos que foram disponibilizados como serviços guardando-os numa lista, lista essa que será sempre usada quando necessário.

De seguida gera-se os wsdl:types: para cada método é verificado se recebe argumentos, qual o tipo de cada argumento e que tipo de resposta esse método devolve, gerando as linhas wsdl apropriadas (nesta fase apenas eram reconhecidos os argumentos do tipo "int", "string" e tipo complexo, sendo que no caso do retorno, o tipo "Void" era adicionado).

Finalizados os wsdl:types, passamos ao wsdl:message, uma mensagem de *SoapIn* e outra de *SoapOut* para cada método. O mesmo se passando em wsdl:portType.

Em wsdl:binding deve existir um para *soap* e outro para *soap12*, não variando muito na estrutura apenas no nome do atributo (*soap* e *soap12*).

Por fim em wsdl:service apenas era necessário saber onde o serviço estaria disponível, sendo para isso introduzido um URL que identificava a localização onde o serviço estaria disponível.

Após a criação deste WSDL, o cliente conseguiu gerar a sua proxy e através dela invocar o web service com as mesmas características do ficheiro *dll*.

Desta forma foi possível criar de raiz um WSDL que pudesse ser disponibilizado para um cliente e este usá-lo na criação de uma proxy para que conseguisse invocar os mesmos métodos identificados pelo WSDL.

3.3.5 5ª Prova: Uso do "appDomain"

Com esta prova de conceito, pretendia-se compreender a forma de funcionamento da classe *appDomain* e como esta poderia ser usada no contexto do Service Broker.

A classe *appDomain* faz parte da Framework NET da Microsoft [4] desde a sua versão 1.0 e esta caracteriza o domínio de uma aplicação, que representa um ambiente isolado onde se podem executar aplicações.

Se uma determinada *assembly* for carregada no domínio principal de uma determinada aplicação, esta não poderá ser libertada de memória enquanto o processo da aplicação estiver a decorrer (executar em memória). A única forma de libertar essa *assembly* será com a paragem da execução da aplicação. No entanto se se criar um domínio aplicacional auxiliar onde será carregada e executada a *assembly*, essa *assembly* poderá ser libertada quando o domínio auxiliar for libertado.

A figura 12 mostra de uma forma simples a ideia anteriormente descrita.

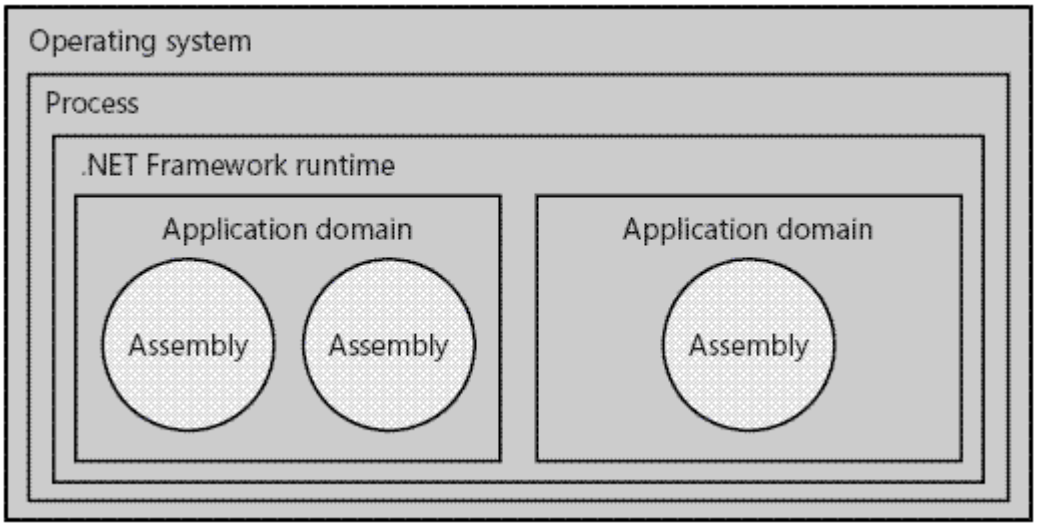


Figura 12: Funcionamento de AppDomains

Este conceito será usado no âmbito do Service Broker, devido à necessidade que este terá de usar e actualizar *dlls* que identificam serviços.

Assumindo que um determinado serviço é identificado por uma *assembly*, para proceder à sua execução, esta deverá ser carregada para memória e posteriormente acedida. Uma vez tendo sido esta *assembly* colocada em memória, a sua libertação está dependente do domínio que a carregou, já que não é possível libertar uma *assembly* especifica de um domínio, estando esta associada ao domínio em si, tendo exactamente o mesmo tempo de vida do domínio onde foi carregada.

A forma de contornar o problema passa por não carregar a *assembly* no domínio criado por defeito pelo processo, mas sim num outro domínio auxiliar. Desta forma a *assembly* será carregada e executada nesse domínio auxiliar e sempre que se pretender proceder a uma actualização da *assembly* previamente carregada, por modificação do serviço por exemplo, apenas será necessário libertar o domínio onde essa *assembly* foi carregada, mantendo a execução do domínio principal (*default*) e consequentemente a execução da aplicação principal. Desta forma, e se for criado um novo domínio por cada *assembly* que se pretenda carregar/usar, é possível de forma virtual efectuar o carregamento e libertação de *assemblys* singulares (libertar cada *assembly* carregada em memória).

Após perceber estes conceitos, passou-se à implementação de um exemplo que pudesse posteriormente ser estendido aquando da implementação do Service Broker.

Assim a prova consistiu na criação de uma classe *Service*, classe essa que identificava um serviço do tipo *dll*. Um serviço era assim caracterizado por um domínio, criado na altura e identificado com o nome "AppDomainID" (onde ID era atribuído aleatoriamente), identificando univocamente um domínio, e por um objecto do tipo *RemoteLibrary*. A existência deste objecto permitia carregar para o domínio auxiliar qualquer *dll* e fornecia um conjunto de operações possíveis de serem executadas sobre essa *dll*. Essas operações consistiam na geração de um WSDL para a *dll* e na invocação de um dos seus métodos, tudo por "*reflection*" (operações disponibilizadas por *RemoteLibrary*).

Por sua vez, o serviço, para além das operações disponibilizadas por *RemoteLibrary* dava também a hipótese de "destruição do serviço", libertando tanto o domínio auxiliar como a *dll* nele carregada.

Esta solução é representada de forma gráfica pela figura 13.

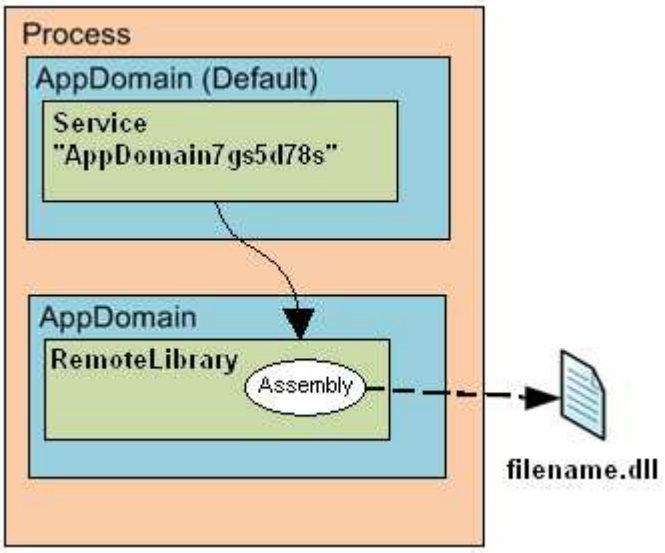


Figura 13: Uso de domínios e carregamento de assembly

Recorrendo a esta forma aparentemente simples de criação, uso e destruição de domínios, foi possível carregar *dlls*, executar operações sobre as mesmas e alterá-las (destruindo os domínios onde estas tinham sido carregadas e repetindo o processo de

criação de domínio e carregamento de *dll*) sem que o domínio principal tivesse alguma vez parado a sua execução.

Por vezes a solução encontrada para a resolução de um determinado problema, esconde as dificuldades encontradas na obtenção da mesma, particularmente se essa solução se apresenta como simples. Foi o caso desta. O total desconhecimento sobre domínios aplicacionais, seu funcionamento, forma de implementação e teoria por detrás da prática (existiam exemplos mas pouca informação do que faziam ou como funcionavam), levou a que se perdesse algum tempo com este componente, o que, uma vez concluída, apenas tornou a sua correcta implementação mais gratificante. Grande parte do problema deveu-se aos exemplos algo simplistas que se encontravam nas várias pesquisas efectuadas. Estes apenas contemplavam a execução de determinada acção nos domínios auxiliares e nunca a transferência de informação/objectos entre domínios ou a invocação de um método desconhecendo por completo o conteúdo da *dll* carregada.

No que diz respeito à passagem de objectos entre domínios, descobriu-se que apenas é possível passar objectos primitivos entre domínios diferentes, todos os outros devem ser do conhecimento de ambos os domínios e estes devem ser serializáveis. Como era necessário enviar o WSDL gerado no domínio auxiliar para o domínio principal, tornou-se necessário criar um objecto *XmlDocument* serializável (Classe que deriva de *XmlDocument* e que implementa a interface *ISerializable*), garantindo assim que o WSDL gerado pudesse circular entre domínios.

Para a invocação de métodos foi usada a técnica de invocação de métodos por reflexão, agora aplicada em domínios auxiliares (usado o método *invoke* descrito na 1ª prova de conceito). Como o objecto que daí resultava era, nesta fase de exploração, sempre primitivo (inteiro ou string), foi sempre possível a circulação do resultado das invocações entre domínios.

Capítulo 4

Desenvolvimento e Implementação do Service Broker

Este capítulo fornecerá uma visão abrangente de todo o projecto, não se focando na forma como cada objectivo foi alcançado mas sim nas funcionalidades de toda a solução e na forma como estas foram implementadas, garantindo assim a realização dos objectivos identificados.

Neste serão apresentados os vários componentes desenvolvidos, as funcionalidades implementadas e os testes de desempenho efectuados ao protótipo obtido.

4.1 Desenvolvimento

Após o estudo de alguns conceitos e verificada a forma de funcionamento dos seus componentes, foi implementada uma solução com o intuito de responder aos objectivos inicialmente propostos.

Foram desenvolvidos três componentes principais:

- Repositório de dados – desenvolvido em SQL Server 2005 [15] e usado para armazenamento de informação necessária ao funcionamento dos dois componentes seguintes;
- Portal de Acesso – desenvolvido em VB.NET e usando o Microsoft Visual Studio 2008 como ferramenta de desenvolvimento. Este componente permite introduzir, alterar e eliminar informação no repositório de dados, informação essa que será usada pelo mesmo portal de acesso, para a apresentar ao utilizador, e pelo Service Broker;

- Service Broker – desenvolvido em C# e usando o Microsoft Visual Studio 2008 como ferramenta de desenvolvimento. Este componente faz uso da informação guardada no repositório de dados para saber de que forma determinado serviço publicado pode ser invocado.

A descrição pormenorizada da implementação destes componentes foi removida, por esta ser uma versão pública do relatório final.

4.2 Testes Efectuados e Resultados Obtidos

Após a construção da primeira versão do Service Broker, e de forma a se perceber quais os custos associados à introdução desta nova camada entre um cliente e o serviço a que pretende aceder, foram efectuados testes de carga a ambos os tipos de serviço que este permite disponibilizar: WebServices e DLL.

Pretendia-se assim conhecer o tempo necessário para efectuar uma invocação através deste novo componente.

4.2.1 Testes e registos

Parar efectuar estes testes, foi usado o sistema para o efeito disponível pela empresa de acolhimento. Este sistema é constituído por um conjunto de máquinas virtuais criadas pelo software/máquina de virtualização *VMware* [10]. Com recurso a este software de virtualização é possível criar um ou mais sistemas operativos simultaneamente num ambiente isolado, criando computadores completos (virtuais) a correr dentro de um computador físico que pode executar um sistema operativo totalmente distinto.

A solução de testes adoptada pela empresa visa a uma tentativa de proporcionar um conjunto de máquinas onde se podem efectuar testes de carga ou de desempenho de aplicações sem a necessidade de despender fundos adicionais na adquirição de novo hardware para novas máquinas.

O sistema operativo usado neste teste em particular foi o Windows Server 2008 R2 para o caso da máquina virtual identificada com o IP 10.0.0.206 onde estavam publicados tanto o Service Broker como o portal de acesso. Também nesta máquina foi publicado um web service simples (apenas efectuava uma conta e retornava a resposta) de forma a comparar o tempo de resposta do web service publicado com o tempo de resposta do mesmo web service mas agora disponível através do Service Broker.

O registo de dados foi processado na máquina local, onde estava instalado o Windows 7 Enterprise N, e foi criado para o efeito uma aplicação simples que simulava o envio de vários pedidos (configurável desde 1 a 1000 pedidos) e registava o tempo que cada pedido demorava a ser satisfeito.

Dois tipos de testes foram executados: testes aos serviços do tipo WebServices e testes aos serviços do tipo DLL.

Serviços do tipo WebServices

Começou-se por medir o tempo de resposta do web service a 50 pedidos sem interposição do Service Broker. Posteriormente foram efectuados pedidos ao mesmo web service, mas publicado pelo Service Broker. Foram efectuados testes desde os 5 pedidos aos 1000 pedidos de forma a ser possível efectuar uma comparação de tempos.

Os tempos registados pela aplicação de testes foram usados para gerar um conjunto de gráficos que mostram o tempo de execução (em segundos) de cada pedido efectuado, ou seja, o intervalo de tempo decorrido desde que a aplicação efectua o pedido até que esta recebe uma resposta ao mesmo.

O teste cujo gráfico de tempos pode ser consultado na figura 14, consistiu numa invocação directa ao web service sendo registado o tempo que este levava a responder. Foi simulada a execução de 50 pedidos em sequência.

No teste efectuado foi registado um tempo de resposta na ordem dos 0,1 segundos numa invocação directa ao web service publicado.

Figura 14: Tempos de resposta de um web service sem interposição do Service Broker. Teste efectuado com 50 pedidos.

Os testes seguintes foram efectuados ao mesmo web service mas desta vez estando este publicado através do Service Broker, sendo a invocação efectuada com recurso ao mesmo.

De forma a medir o desempenho do Service Broker em diferentes cenários de carga, foram realizados testes com 5, 10, 20 e 1000 pedidos, sendo registados os seus tempos de resposta.

O gráfico da figura 15 mostra os tempos de resposta obtidos em 5 testes realizados, cada um com 5 pedidos. Constatou-se, com a realização destes testes, que o primeiro pedido demora sempre mais tempo a ser satisfeito que os restantes, e que a cada novo teste (sendo todos executados nas mesmas circunstâncias) existe um ligeiro aumento no tempo de resposta.

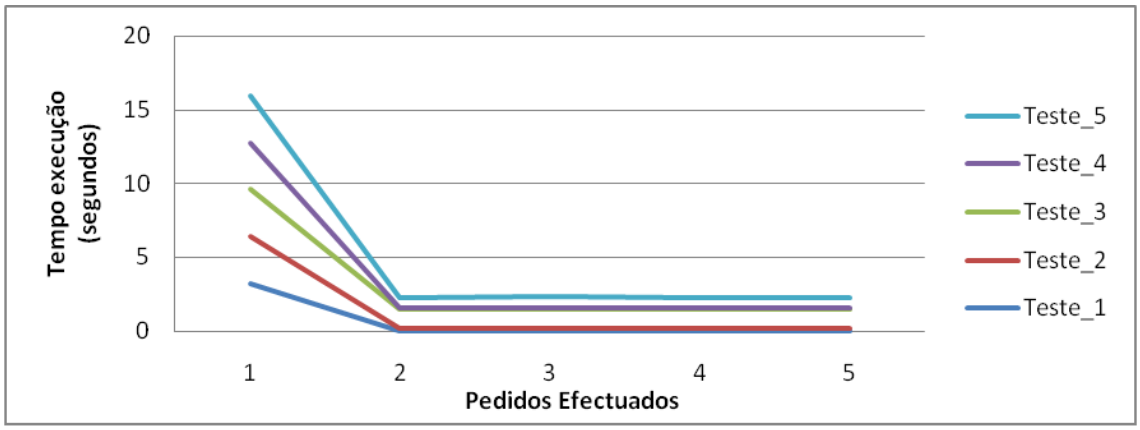


Figura 15: Tempos de resposta de um web service publicado através do Service Broker. Efectuados 5 testes cada um com 5 pedidos

O gráfico da figura 16 apresenta os resultados obtidos nos 4 testes efectuados, cada um com 10 pedidos. Este apresenta um comportamento semelhante aos resultados mostrados pela figura 15 com uma ligeira diferença, o aumento do número de pedidos efectuados parece ter aumentado o tempo de resposta a esses pedidos, notando-se ainda mais essa perda de desempenho a partir do 6 pedido efectuado.

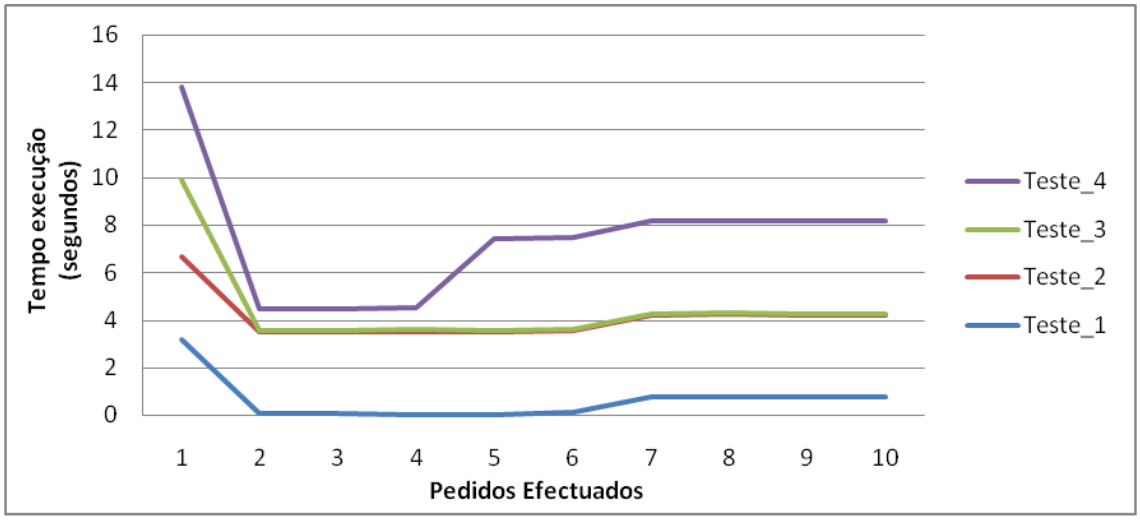


Figura 16:Tempos de resposta de um web service publicado através do Service Broker. Efectuados 3 testes cada um com 10 pedidos

O gráfico da figura 17 mostra os tempos de resposta obtidos nos 3 testes efectuados, sendo que desta vez cada teste era composto por um conjunto de 20 pedidos

cada. Mais uma vez se nota que o primeiro pedido demora um pouco a ser satisfeito e que a cada novo teste existe um aumento no tempo de resposta por pedido, à semelhança daquilo que já acontecia nos gráficos anteriores. No entanto quando se efectuaram 20 pedidos de seguida, o tempo de resposta a partir do quinto pedido aumentou consideravelmente (aumentando ainda mais a cada novo teste).

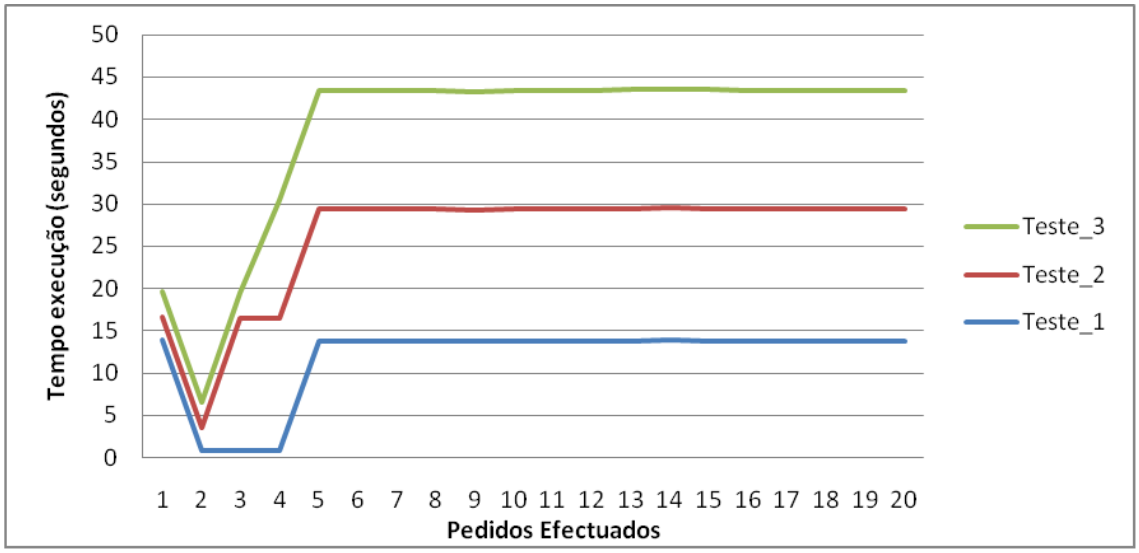


Figura 17: Tempos de resposta de um web service publicado através do Service Broker. Efectuados 3 testes cada um com 20 pedidos

Foi ainda efectuado um teste final com 1000 pedidos de forma a medir a capacidade de resposta do Service Broker durante um período de elevada carga, sendo os seus resultados mostrados pelo gráfico da figura 18. Este teste apresentou um comportamento semelhante aos testes anteriores com a diferença de que neste, e devido ao seu elevado número de pedidos, foi possível verificar uma ligeira melhoria do tempo de resposta à medida que os pedidos iam sendo satisfeitos.

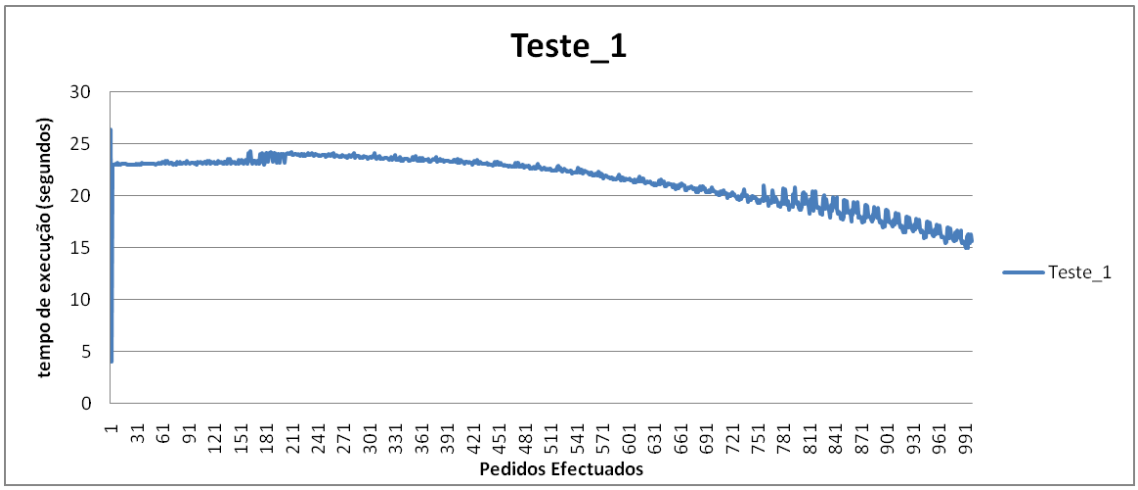


Figura 18: Tempos de resposta de um web service publicado através do Service Broker. Efectuado um teste com 1000 pedidos

Em todos os testes efectuados (excepção feita para o último teste já que apenas foi realizado um teste com 1000 pedidos) é possível verificar que a cada teste existe uma ligeira perda de desempenho em relação ao teste anterior, isto pode ter uma de duas razões; ou a capacidade da máquina que está a registar os tempos, perde desempenho no registo, ou existe uma ligeira perda de desempenho ao nível do Service Broker no que respeita à sua capacidade de resposta na satisfação de vários pedidos em simultâneo.

Em todos os gráficos verifica-se que o primeiro pedido efectuado demora sempre mais tempo que os seguintes, e que a perda de desempenho começa a ser mais acentuada quando são efectuados mais de 10 pedidos em simultâneo, mas se o número de pedidos não ultrapassar muito o numero referido, o tempo de resposta varia entre 1 e 4 segundos por resposta.

No teste onde foram efectuados 1000 pedidos, o tempo de resposta começa por ser algo elevado, sendo este na ordem dos 22 ou 23 segundos por resposta, no entanto este tempo tende a diminuir à medida que os pedidos vão sendo satisfeitos. A partir do pedido número 300 ou 350 nota-se uma ligeira descida no tempo de resposta, tendo sido registado um tempo de 15 segundos para o último pedido.

Serviços do tipo DLL

Uma vez que o Service Broker desenvolvido permite a publicação de dois tipos de Serviços, WebServices e DLL, o próximo teste baseou-se numa forma de medir o desempenho de uma invocação feita directamente a uma *dll* usando a técnica de reflexão (*reflection*) e uma invocação onde era usado o Service Broker para fazer essa invocação.

Era sabido à partida que uma invocação recorrendo ao Service Broker demoraria sempre mais tempo, quanto mais não seja devido ao facto de a invocação feita via Service Broker, ter de recorrer à transmissão de dados via rede, o que não acontece numa invocação executada directamente à *dll*. No entanto seria sempre interessante saber quanto mais tempo levaria.

De forma a medir e comparar o tempo dispendido numa invocação de um serviço do tipo DLL, foi criada uma classe com o método `Somar` que efectua a soma entre dois inteiros. Esta foi compilada, sendo a sua *dll* resultante acedida localmente usando reflexão (*reflection*) (invocando o seu método) e publicada como um serviço no Service Broker. Para efectuar o teste foi utilizada a mesma aplicação usada para testar os serviços do tipo WebService, efectuando 500 pedidos para o mesmo método e medido o seu tempo de resposta.

O gráfico da figura 19 mostra os tempos de execução obtidos. A invocação por reflexão foi feita localmente à *dll* e os valores obtidos foram, como seria de esperar, surpreendentemente rápidos. Assim a invocação mais rápida demorou apenas 0,001 segundos, a mais lenta demorou 0,071 segundos, com um tempo médio por invocação de 0,002 segundos.

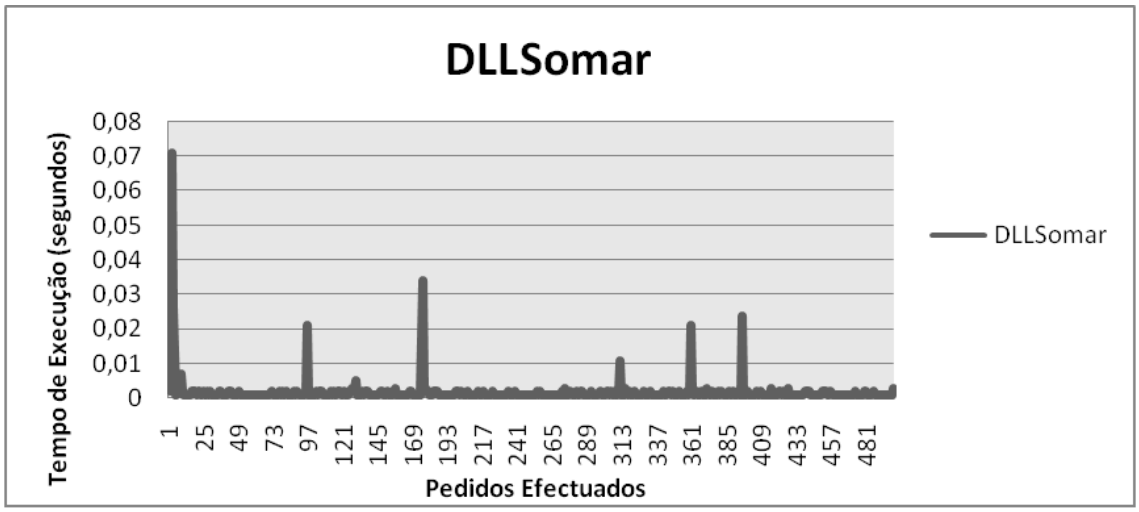


Figura 19: Tempo de resposta de uma invocação directa à dll usando reflexão. Teste efectuado com 500 pedidos.

Após a invocação directa procedeu-se à publicação da *dll* pelo Service Broker. Foram também efectuados 500 pedidos e os seus tempos de resposta foram registados, sendo os mesmos apresentados no gráfico da figura 20.

Como seria de esperar, uma invocação através do Service Broker demorou muito mais tempo do que a invocação directa por reflexão (*reflection*). A invocação mais rápida foi registada com o tempo de 4,94 segundos, a mais lenta demorou 87,28 segundos, sendo o tempo médio de resposta de uma invocação de 33,25 segundos.

A invocação a um serviço do tipo DLL registou no entanto um comportamento diferente de uma invocação do tipo WebService, para além desta demorar mais tempo a ser executada foi registado um aumento no tempo de resposta a cada novo pedido efectuado, sendo este aumento considerável a partir do pedido 300.

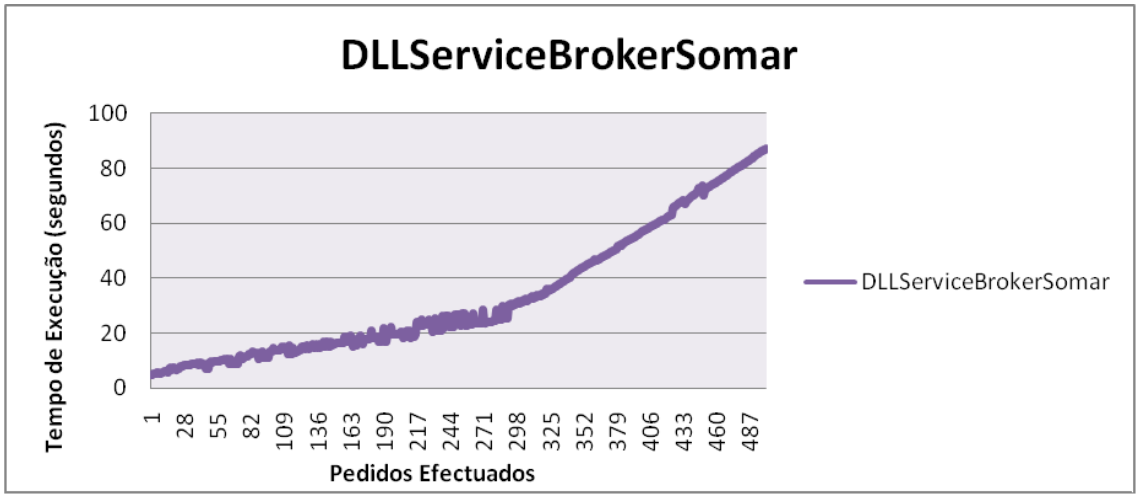


Figura 20: Tempo de resposta de uma invocação à dll publicada pelo Service Broker. Teste efectuado com 500 pedidos.

4.2.2 Resumo dos resultados obtidos

Nos testes efectuados foi notório o aumento do tempo de resposta por parte do web service quando este estava publicado pelo Service Broker. Este respondia aos pedidos em cerca de 0,1 segundos numa invocação directa e entre 1 a 4 segundos numa invocação pelo Service Broker.

De notar que à medida que o número de pedidos ia aumentando também o tempo de resposta de cada pedido ia crescendo, não ficando claro se este tipo de situação se devia à perda da capacidade de resposta por parte da máquina local (que registava os tempos), se da máquina onde o Service Broker estava alojado (uma vez que esta se tratava de uma máquina virtual) ou se se tratava de uma incapacidade de resposta por parte do Service Broker perante um número elevado de pedidos. Ficando evidente uma notória deterioração na capacidade de resposta a partir de 10 pedidos.

Quanto aos serviços do tipo DLL estes revelaram um comportamento um pouco diferente do registado para os serviços do tipo WebService. Era de esperar que estes tivessem um tempo de resposta mais alto do que os web services acessíveis pelo Service Broker, devido à quantidade de operações efectuadas para realizar uma invocação, faltando saber quão mais tempo seria na realidade (em média). Notou-se um declínio considerável no desempenho à medida que o número de pedidos ia aumentando, não sendo mais uma vez muito claro do porquê desse comportamento tendo como possíveis

motivos os já apresentados para os serviços do tipo WebService. No entanto se o número de pedidos não ultrapassa-se os 50 pedidos poder-se-ia esperar uma resposta na ordem dos 10 segundos.

Era sabido à partida que a introdução de um ponto intermédio iria sempre aumentar o tempo de resposta a um pedido. No final interessa fazer um balanço e ponderar se as vantagens introduzidas por este tipo de componente, permitem compensar e suportar o tempo de resposta acrescentado.

Capítulo 5

Conclusões

5.1 Objectivos Realizados

Pretendia-se com esta proposta, a elaboração de um Service Broker que actuasse como um intermediário entre o utilizador e os serviços a que este pretendesse aceder, ou seja, que criasse uma camada adicional escondendo a camada de lógica de negócio da camada de interface com o utilizador.

Pretendia-se ainda que fosse criado um portal de acesso e de administração, para que o administrador pudesse proceder à configuração do Service Broker desenvolvido e para que o utilizador comum pudesse consultar os seus serviços.

Por fim poder-se -ia ser estudada uma forma de estender o tipo de comunicação usada entre o utilizador e o Service Broker para além do SOAP. Devendo para isso ser estudada e compreendida a Framework WCF da Microsoft usada para construir aplicações SOA permitindo o envio de dados de forma assíncrona de um ponto para o outro.

No que diz respeito ao desenvolvimento do Service Broker, a sua implementação consistiu numa forma de disponibilizar dois tipos de serviços: um web service e uma *assembly*. O web service foi a forma de serviço mais simples de disponibilizar, uma vez que este já é por si só um serviço pronto para ser acedido, sendo apenas da responsabilidade do Service Broker a sua ocultação, adição de uma camada de segurança efectuando controlo de acesso aos serviços e reenvio de mensagens entre o utilizador e o web service, ficando esta forma de acesso concluída. No caso da disponibilização de um serviço do tipo *assembly*, este revelou-se de implementação mais complicada uma vez que para disponibilizar uma *assembly* como um serviço, foi necessário realizar várias sub-acções que permitissem a disponibilização das operações

da *assembly* e a sua invocação. A disponibilização de operações é conseguida através da criação de um WSDL contendo todas as operações possíveis e parâmetros a usar na invocação, enquanto que a invocação destas operações é conseguida através do uso de domínios auxiliares e da serialização e desserialização de objectos correspondente à forma de funcionamento do SOAP e da necessidade de passar objectos desconhecidos entre domínios. Não foi possível no entanto estender esta forma de acesso a todos os tipos de objectos, ficando para já apenas funcional para objectos primitivos ou cujo processo de serialização e desserialização seja já oferecido pela biblioteca .NET, o que implica uma conclusão parcial desta forma de acesso.

Foi possível efectuar um elevado número de invocações pelo Service Broker desenvolvido para ambos os tipos de serviços possíveis de publicar, sendo o seu grau de realização elevado no que diz respeito a esta vertente.

No caso do portal de acesso este foi elaborado de forma a ser o mais funcional possível fornecendo as informações necessárias para cada tipo de utilizador. Este desempenha tanto funções de administração como funções de visualização de dados. Neste portal é possível melhorar alguns aspectos na forma como a informação é apresentada ao utilizador, no entanto no geral o seu grau de realização apresenta-se como elevado.

Quanto ao estudo que pretendia estender a forma de comunicação do Service Broker com os seus clientes para além de HTTP+XML, nesta vertente muito pouco ou quase nada foi conseguido. Foram apenas efectuados alguns testes de forma a constatar as potencialidades deste tipo de tecnologia, mas estes testes nunca passaram disso mesmo, sendo o grau de realização deste componente muito baixo.

Em resumo, o objectivo geral de desenvolver e implementar um Service Broker para disponibilização de serviços como web services parece-me ter sido conseguido, tendo ficado disponível um protótipo experimental ao qual foram efectuados alguns testes de desempenho, comprovando assim as potencialidades de um componente deste tipo.

5.2 Outros trabalhos realizados

Durante a realização deste projecto e de forma a satisfazer as necessidades da empresa, foram realizados outros trabalhos paralelos que não faziam parte dos objectivos nem do trabalho principal, mas que no entanto serviram ainda assim para adquirir novos conhecimentos, não só nas ferramentas de desenvolvimento mas também dos métodos de trabalho envolvidos.

Um desses trabalhos envolveu a criação de uma aplicação em C# com recurso à ferramenta de desenvolvimento Visual Studio 2008. Esta permitia efectuar o download, para a máquina local, de ficheiros que se encontravam armazenados remotamente e periodicamente verificava se esses ficheiros remotos tinham sido actualizados ou não, efectuando a actualização dos ficheiros locais em caso afirmativo. Esta aplicação permitia ainda a modificação local desses ficheiros abrindo-os para edição. Outra das suas funcionalidades consistia numa forma simples de configuração por parte do utilizador, como era o caso de quais os ficheiros a fazer download e cuja actualização iria ser verificada, o intervalo de tempo entre verificações e o local onde guardar os ficheiros. Estes dados eram guardadas num ficheiro XML que era posteriormente usado pela aplicação.

A elaboração desta pequena aplicação, a ser incluída no system tray do Windows, serviu como forma de familiarização da estrutura de ficheiros em uso pela empresa, da estrutura usada por um ficheiro XML e forma de construir, usar e manipular ficheiros XML.

Outro trabalho envolveu a criação de recursos multilingue para a aplicação WebDoc 2010 (aplicação de gestão documental em desenvolvimento) e adaptação de código para uso desses recursos.

A participação neste projecto foi importante devido à oportunidade de trabalhar em equipa e numa situação em que múltiplos utilizadores estavam a aceder ao mesmo tempo ao mesmo projecto. Este trabalho permitiu perceber as verdadeiras vantagens de trabalhar com o TFS (Team Foundation Server), uma ferramenta indispensável para a elaboração de projectos de desenvolvimento de software de forma cooperativa, já que esta oferece não só um repositório comum de código onde todos podem aceder, mas também permite controlar versões de código submetidas, limitar o número de pessoas que acedem a determinado ficheiro, entre outras.

Por fim a oportunidade de trabalhar no acrescento de dois módulos na versão base do Portal da Câmara Municipal de Ponta Delgada. Este trabalho consistiu maioritariamente na construção de um modelo de dados para cada um dos módulos que fosse capaz de armazenar toda a informação relevante para o mesmo e na construção de dois tipos de componentes: um Back-End e um Front-End. Uma forma simples de explicar este dois componentes é de que o Back-End é responsável pela introdução de informação no modelo de dados criado e o Front-End é responsável por apresentar a informação armazenada.

Este trabalho revelou-se bastante desafiante, não só porque não existia há partida uma ideia clara do tipo de informação que seria necessária armazenar para ambos os módulos nem de que forma esta deveria ser apresentada, mas também devido ao desconhecimento de toda a lógica aplicacional por detrás deste portal. Estes factos tornaram o desenvolvimento deste projecto moroso em certas tarefas tornando-se necessária a sua exploração com o objectivo de perceber as funcionalidades já implementadas e de forma a estendê-las de forma coerente. Este não deixou no entanto de contribuir para a adquirição de múltiplos conhecimentos.

5.3 Limitações e trabalho futuro

Como todo o projecto que tem como base o processo exploratório de conceitos, verificando assim a sua exequibilidade para mais tarde ser aplicado num projecto de maiores dimensões, este apresentará sempre limitações e será sempre possível melhorar aspectos não só das funcionalidades já implementadas mas também através da adição de novas, enriquecendo assim o produto final.

Uma das principais limitações deste projecto reside no processo de serialização e desserialização de objectos necessário à invocação de serviços do tipo DLL. No envio/recepção de objectos primitivos (Integer, String) ou de objectos cuja biblioteca .NET possua uma implementação de serialização/desserialização, o processo de invocação é efectuado com sucesso e produz o resultado esperado. No entanto, no caso dos objectos mais complexos, como acontece com os objectos definidos pelo utilizador, o processo de invocação falha porque o Service Broker não sabe lidar com este tipo de objecto.

Por outro lado, a geração de WSDL para *assemblies* encontra-se de certa forma limitada porque esta baseia-se em certos pressupostos, como é o caso de objectos primitivos e outros não primitivos mas onde foi possível extrapolar que tipo de informação deveria constatar num WSDL de forma a identificar esse tipo de objectos.

As duas limitações anteriormente mencionadas, encontram-se directamente relacionadas, uma vez que ao saber gerar a informação de como serializar um objecto e ao disponibilizá-la no WSDL, deveria ser possível efectuar a sua serialização aquando do seu envio e desserialização na sua recepção.

Uma outra limitação, é o facto de se desconhecer qual o comportamento obtido na invocação de um serviço do tipo DLL quando a *assembly* que o identifica depende de outra (porque usa uma função de outra *assembly* por exemplo). Como este tipo de invocação ocorre num domínio diferente do domínio por defeito, e uma vez que estes domínios apenas reconhecem as *assemblys* que lhes são explicitamente carregadas, é espectável que caso uma *assembly* dependa de outra, que não tenha sido carregada no domínio onde a invocação é efectuada, possa resultar numa invocação falhada.

Outra das limitações deste trabalho reside no facto de em algumas situações não ser dada qualquer informação ao utilizador do motivo porque determinada operação não foi executada. Existindo assim a possibilidade de alguns melhoramentos ao nível das mensagens de erro a serem criadas e enviadas ao utilizador, de forma a fornecer mais feedback aplicacional.

Como trabalho futuro interessa assinalar a capacidade de se gerar um WSDL para *assemblies* de forma mais dinâmica do que aquela que foi implementada. Esta deveria fazer uso de XMLSchemas para gerar a estrutura das mensagens a serem geradas pelo cliente e talvez assim fosse possível ultrapassar tanto a limitação de não serem geradas mensagens para os objectos complexos, como uma forma de ser possível efectuar a serialização/desserialização de objectos complexos.

Ainda como trabalho futuro e com o intuito de introduzir novas capacidades no Service Broker desenvolvido, aumentando assim a sua usabilidade, importa referir a inclusão de WCF de forma a estender a sua capacidade de comunicação para além de HTTP+XML (SOAP) e na inserção de uma forma de controlo sob o número de invocações permitidas por cada utilizador a cada serviço ou a cada método de um serviço. Esta última solução permitiria a utilização do Service Broker como um *SaaS*

(System as a Service) numa abordagem do género *pay-per-use* onde cada utilizador poderia pré-comprar um determinado número de invocações para um serviço, ou onde o serviço era invocado um número de vezes por um utilizador e no final lhe era apresentado o valor total a pagar.

5.4 Apreciação Final

A realização deste projecto permitiu pôr à prova todos os conhecimentos adquiridos durante a frequência do curso de Engenharia Informática, além de contribuir para uma reformulação intelectual e pessoal. Este contacto com o mundo empresarial permitiu adquirir novos conhecimento não só ao nível técnico, necessário para a vida de um futuro engenheiro informático, mas também ao nível das chamadas *softskills*, oferecendo uma visão do mercado de trabalho muito diferente daquela que se obtém durante a frequência de um curso superior.

Existiram grandes diferenças entre a realização deste projecto e outros realizados durante a frequência do curso. Para além do contexto e dimensão, o facto de este ter sido realizado a solo (sem possibilidade de discutir formas concretas de implementação), aliado ao facto de não possuir qualquer tipo de documentação especificando os seus requisitos, contribuiu para que houvesse uma estranheza inicial. No entanto a sua componente de investigação e procura por novas formas de alcançar determinados objectivos, tornou este projecto desafiante e acabou por se revelar bastante gratificante, sempre que era encontrada uma solução para resolver determinado problema.

No final penso que a solução desenvolvida possa trazer uma mais-valia para a empresa, na medida em que permite que esta tenha um maior controlo sobre os serviços desenvolvidos e que queira disponibilizar para acesso exterior. Ao “esconder” estes serviços e ao limitar a disponibilização dos mesmos apenas a utilizadores autorizados, é introduzida uma barreira que oferece alguma segurança e que deve ser ultrapassada por qualquer utilizador mal intencionado, oferecendo assim maior segurança e controlo sobre os seus serviços.

Quanto à vertente em que uma *assembly* é disponibilizada como um web service, esta poderá permitir à empresa a disponibilização de *assemblies* anteriormente

desenvolvidas, como serviços possíveis de serem invocados, sem que estas tenham sido pensadas para esse tipo de acesso na altura da sua criação. Desta forma, em vez de ser criado um novo web service que basicamente iria ter o mesmo conteúdo da *assembly* que se queria expôr, uma nova forma de disponibilização é deixada a cargo do Service Broker, ficando este responsável pela geração de toda a informação necessária para disponibilizar esse conteúdo para invocação, automatizando e dinamizando as operações a efectuar.

Penso que as vantagens que este tipo de componente possa trazer a uma empresa, se sobrepõem ao facto de existir uma perda de desempenho nos serviços disponibilizados pelo mesmo. Com a constante evolução e adição de novas funcionalidades a este tipo de componente, considero que essas vantagens ganharão ainda mais peso, minimizando de forma considerável as falhas deste.

Bibliografia

[1] Redler, Rickard; Rossberg, Joachim "*Pro Scalable .NET 2.0 Application Designs*", Apress, 2006, 207-276

[2] J.D. Meier; Alex Homer; David Hill; Jason Talor, "*Application Architecture Guide 2nd Edition (Patterns & Practices)*", Microsoft, 2008

[3] AMBISIG, S.A., "*Manual do Sistema de Gestão Integrado*" v7, 2008, Lisboa, 10-11

[4] MSDN, *Library* (página online disponível em http://msdn.microsoft.com/en-us/library contendo toda a informação para desenvolver produtos baseados em tecnologia Microsoft), [10.6.10]

[5] Wikipedia, *Extreme Programming (XP)* http://en.wikipedia.org/wiki/Extreme_programing, [22.6.10]

[6] Wikipedia, *.NET assembly*

http://en.wikipedia.org/wiki/.NET_assembly, [22.6.10]

[7] Wikipedia, *Hypertext Transfer Protocol*

http://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol, [22.6.10]

[8] Wikipedia, *Windows Communication Foundation*
http://en.wikipedia.org/wiki/Windows_Communication_Foundation, [22.6.10]

[9] Wikipedia, *Service Oriented Architecture*
http://pt.wikipedia.org/wiki/Service-oriented_architecture, [22.6.10]

[10] Wikipedia, *VMware*
http://pt.wikipedia.org/wiki/VMware, [23.6.10]

[11] The World Wide Web Consortium (W3C), *Web Services Description Language*
http://www.w3.org/TR/wsdl, [22.6.10]

[12] The World Wide Web Consortium (W3C), *eXtensible Markup Language*
http://www.w3.org/TR/xml/, [22.6.10]

[13] The World Wide Web Consortium (W3C), *Simple Object Access Protocol*
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, [22.6.10]

[14] The World Wide Web Consortium (W3C), *Web Services Architecture*
http://www.w3.org/TR/ws-arch/. [5.7.10]

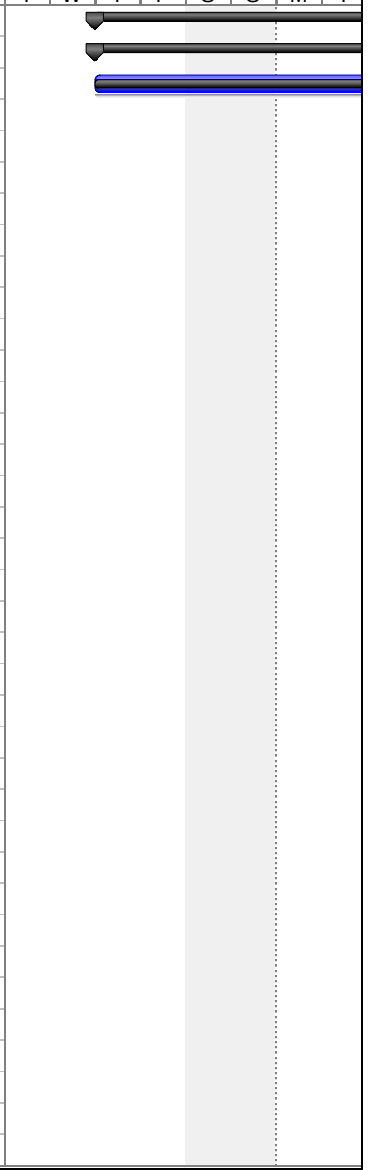
[15] Microsoft, *Microsoft SQL Server 2005*
http://www.microsoft.com/sqlserver/2005/en/us/express.aspx, [24.6.10]

[16] Microsoft, *Microsoft Development, MSDN Subscriptions, Resources and More*
http://msdn.microsoft.com/pt-pt/default.aspx, [24.6.10]

Anexos

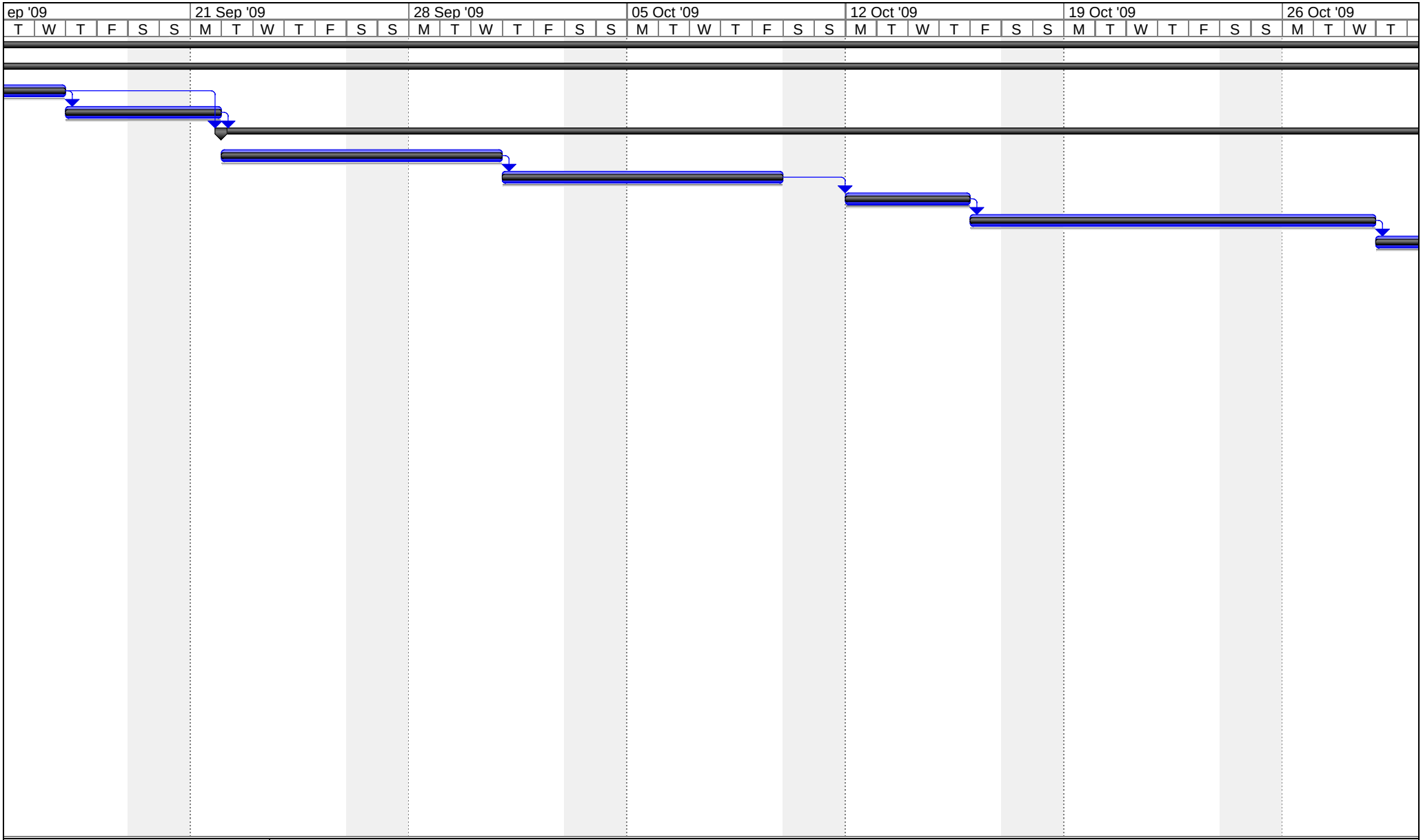
Mapa de Gantt

ID	Task Name	Duration	Start	Finish	Predecessors
1	**Implementação de um Service Broker**	**196 days**	**Thu 10-09-09**	**Thu 10-06-10**	
2	**Fase de Iniciação**	**40 days**	**Thu 10-09-09**	**Wed 04-11-09**	
3	Análise dos requisitos a serem implementados	5 days	Thu 10-09-09	Wed 16-09-09	
4	Contextualização com o ambito da actividade	3 days	Thu 17-09-09	Mon 21-09-09	3
5	**Realização de provas de conceito iniciais**	**32 days**	**Tue 22-09-09**	**Wed 04-11-09**	**3;4**
6	Cliente de WSDL	7 days	Tue 22-09-09	Wed 30-09-09	
7	HTTPHandler para captura de tráfego	7 days	Thu 01-10-09	Fri 09-10-09	6
8	Uso de SOAP Extensions	4 days	Mon 12-10-09	Thu 15-10-09	7
9	Construção Dinâmica de WSDL	9 days	Fri 16-10-09	Wed 28-10-09	8
10	Uso do AppDomain	5 days	Thu 29-10-09	Wed 04-11-09	9
11	Escrita do Relatório Preliminar	5 days	Mon 09-11-09	Fri 13-11-09	
12	**Fase de Execução**	**125 days**	**Thu 05-11-09**	**Wed 28-04-10**	**2**
13	Estudo dos casos de uso	3 days	Thu 05-11-09	Mon 09-11-09	
14	Elaboração de Diagramas de Sequência	7 days	Tue 10-11-09	Wed 18-11-09	13
15	Elaboração de Diagramas de Fluxo	8 days	Thu 19-11-09	Mon 30-11-09	14
16	Implementação da versão base do Service Broker	15 days	Tue 01-12-09	Mon 21-12-09	15
17	Utilização de SOAP Extension e SOAP Fault	4 days	Tue 22-12-09	Fri 25-12-09	16
18	**Criação dinâmica de WSDL para componentes sem WebService**	**10 days**	**Mon 28-12-09**	**Fri 08-01-10**	**17**
19	Forma de usar appDomain	5 days	Mon 28-12-09	Fri 01-01-10	
20	Geração de WSDL em appDomain	5 days	Mon 04-01-10	Fri 08-01-10	19
21	Criação de WebServices de administração	10 days	Mon 11-01-10	Fri 22-01-10	18
22	**Criação de um portal de administração**	**30 days**	**Mon 25-01-10**	**Fri 05-03-10**	**21**
23	Criação de um serviço	10 days	Mon 25-01-10	Fri 05-02-10	
24	Listagem de Serviços	5 days	Mon 08-02-10	Fri 12-02-10	23
25	Visualização dos dados de um Serviço	5 days	Mon 15-02-10	Fri 19-02-10	24
26	Listagem de Métodos	5 days	Mon 22-02-10	Fri 26-02-10	25
27	Formulário de teste a um Método de um Serviço	5 days	Mon 01-03-10	Fri 05-03-10	26
28	Execução de logs de utilização do Serviço	5 days	Mon 08-03-10	Fri 12-03-10	23
29	Implementação de permissões nos métodos disponibilizados	7 days	Mon 15-03-10	Tue 23-03-10	22
30	Implementação de níveis de segurança nos serviços disponibilizados	7 days	Wed 24-03-10	Thu 01-04-10	29
31	Adaptação do portal de administração para uso generalizado	10 days	Fri 02-04-10	Thu 15-04-10	30
32	Invocação de componentes sem WebService	6 days	Fri 16-04-10	Fri 23-04-10	
33	Geração dinâmica de WSDL de acordo com permissões do utilizador	3 days	Mon 26-04-10	Wed 28-04-10	29
34	**Fase de Conclusão**	**8 days**	**Thu 29-04-10**	**Mon 10-05-10**	**12**
35	Benchmarking ao Service Broker	4 days	Thu 29-04-10	Tue 04-05-10	
36	Execução de testes unitários	4 days	Wed 05-05-10	Mon 10-05-10	35
37	Elaboração do Relatório Final	23 days	Tue 11-05-10	Thu 10-06-10	34



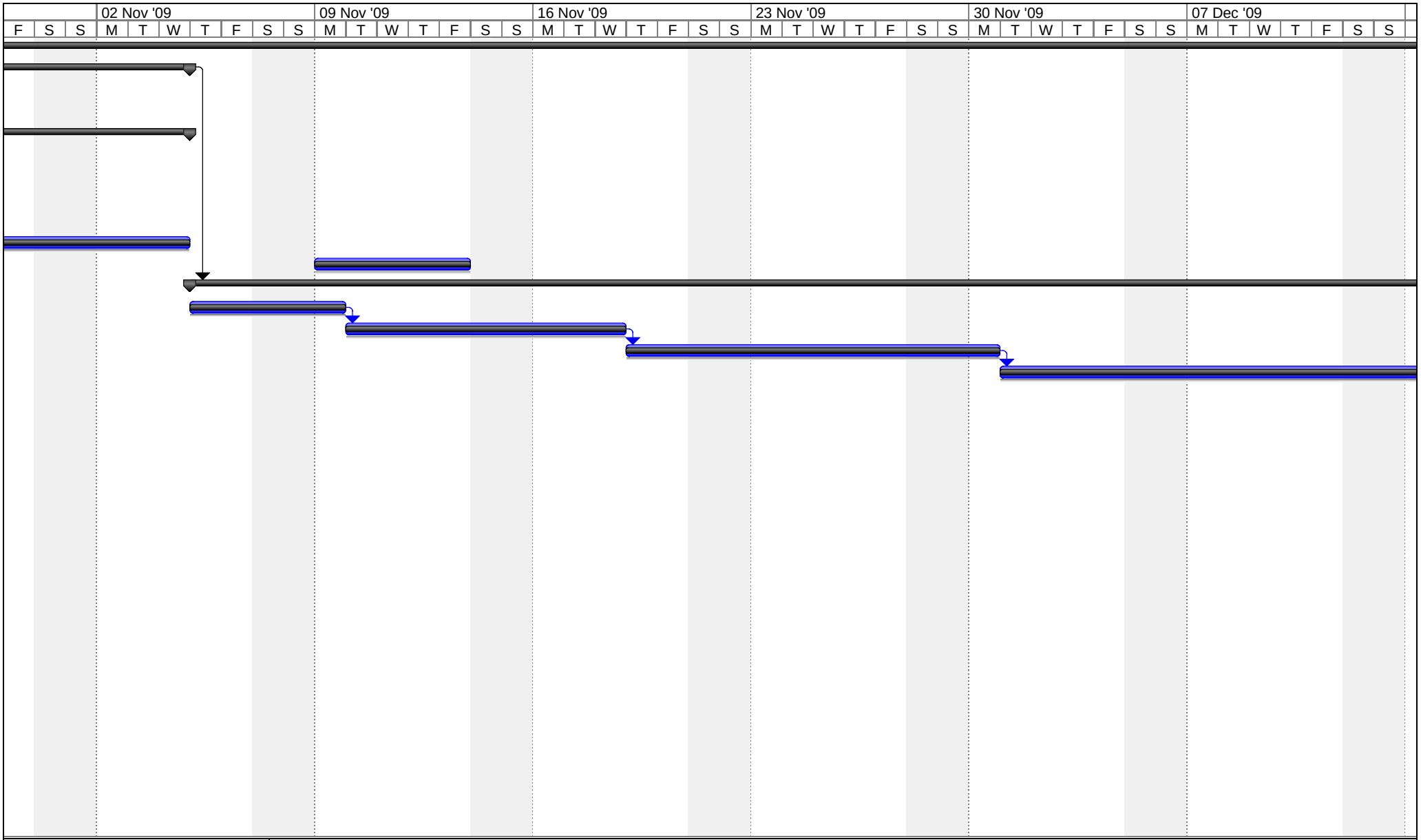
Project: Implementação de um Servic
Date: Fri 09-07-10

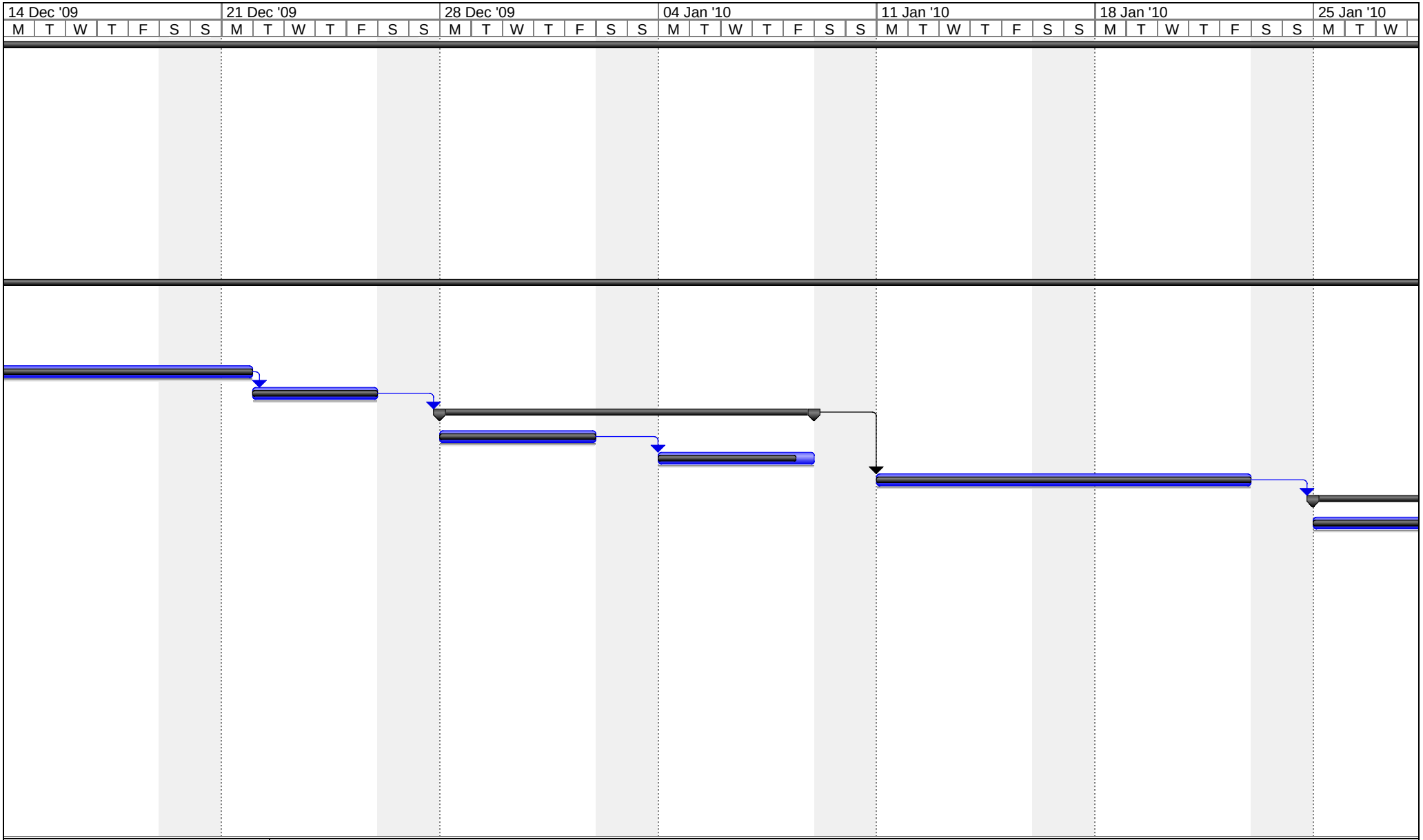
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline



Project: Implementação de um Servic
Date: Fri 09-07-10

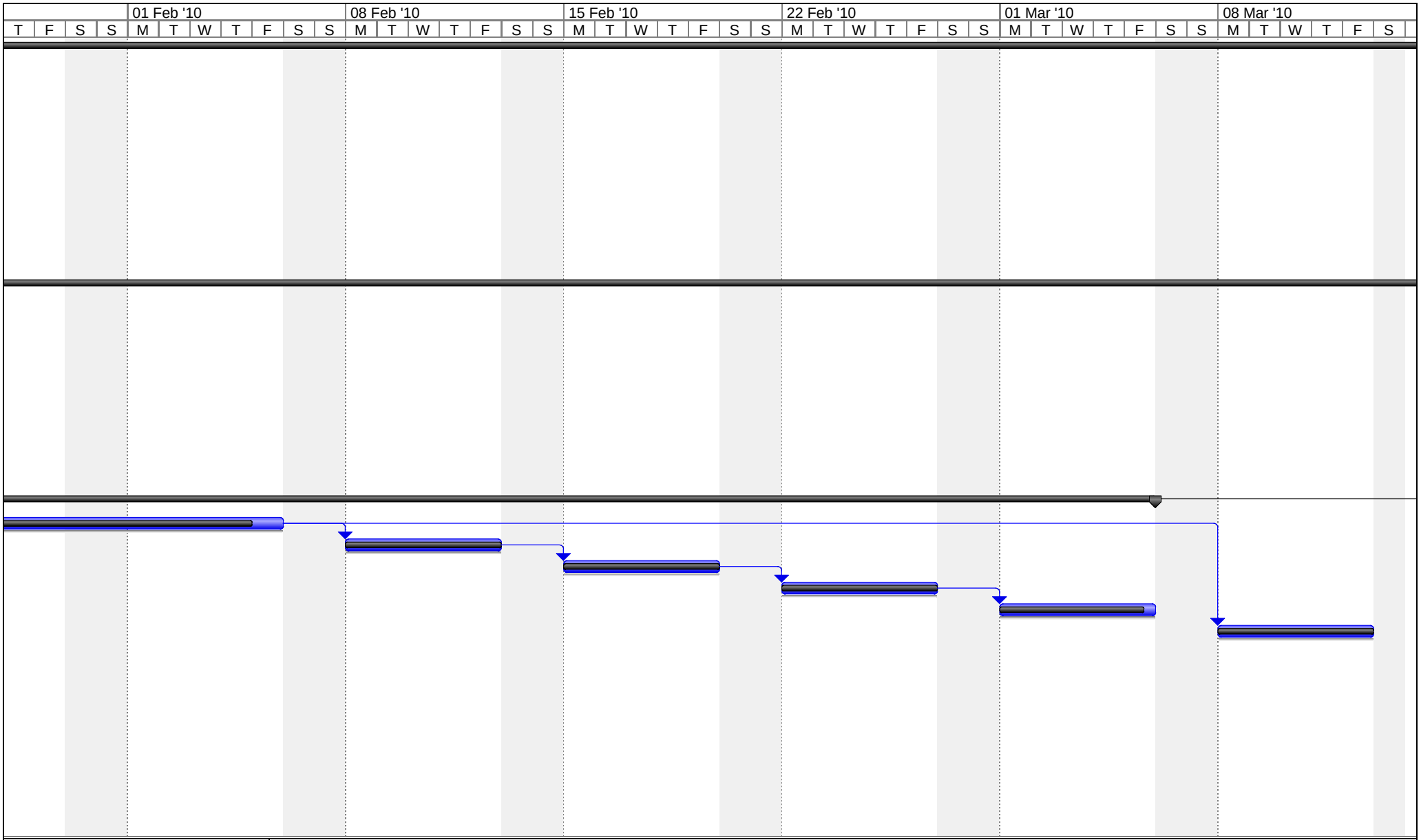
Task		Milestone		External Tasks	
Split		Summary		External Milestone	
Progress		Project Summary		Deadline	

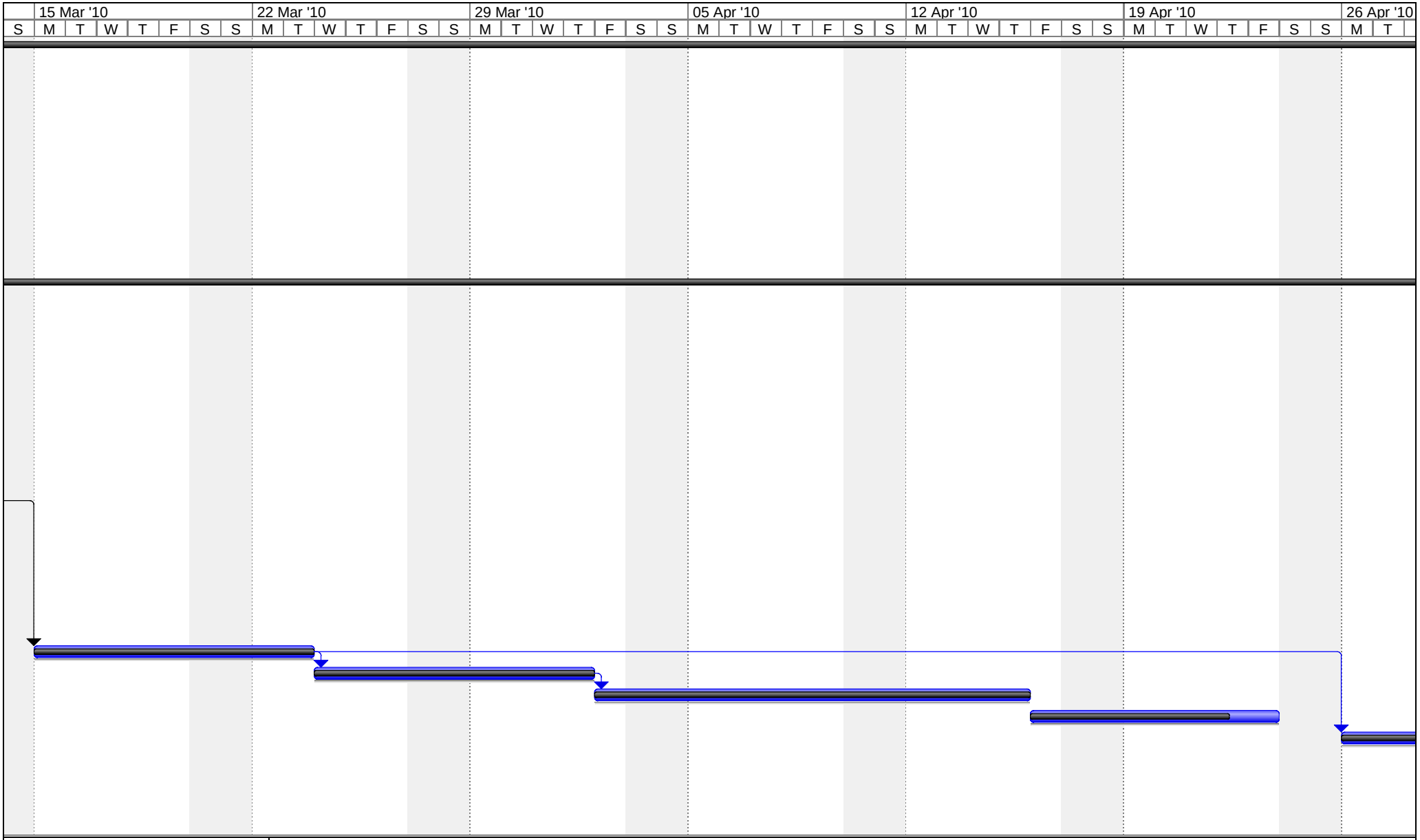

02 Nov '09
09 Nov '09
16 Nov '09
23 Nov '09
30 Nov '09
07 Dec '09
F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S
Project: Implementação de um Servic
Date: Fri 09-07-10
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline



Project: Implementação de um Servic
Date: Fri 09-07-10

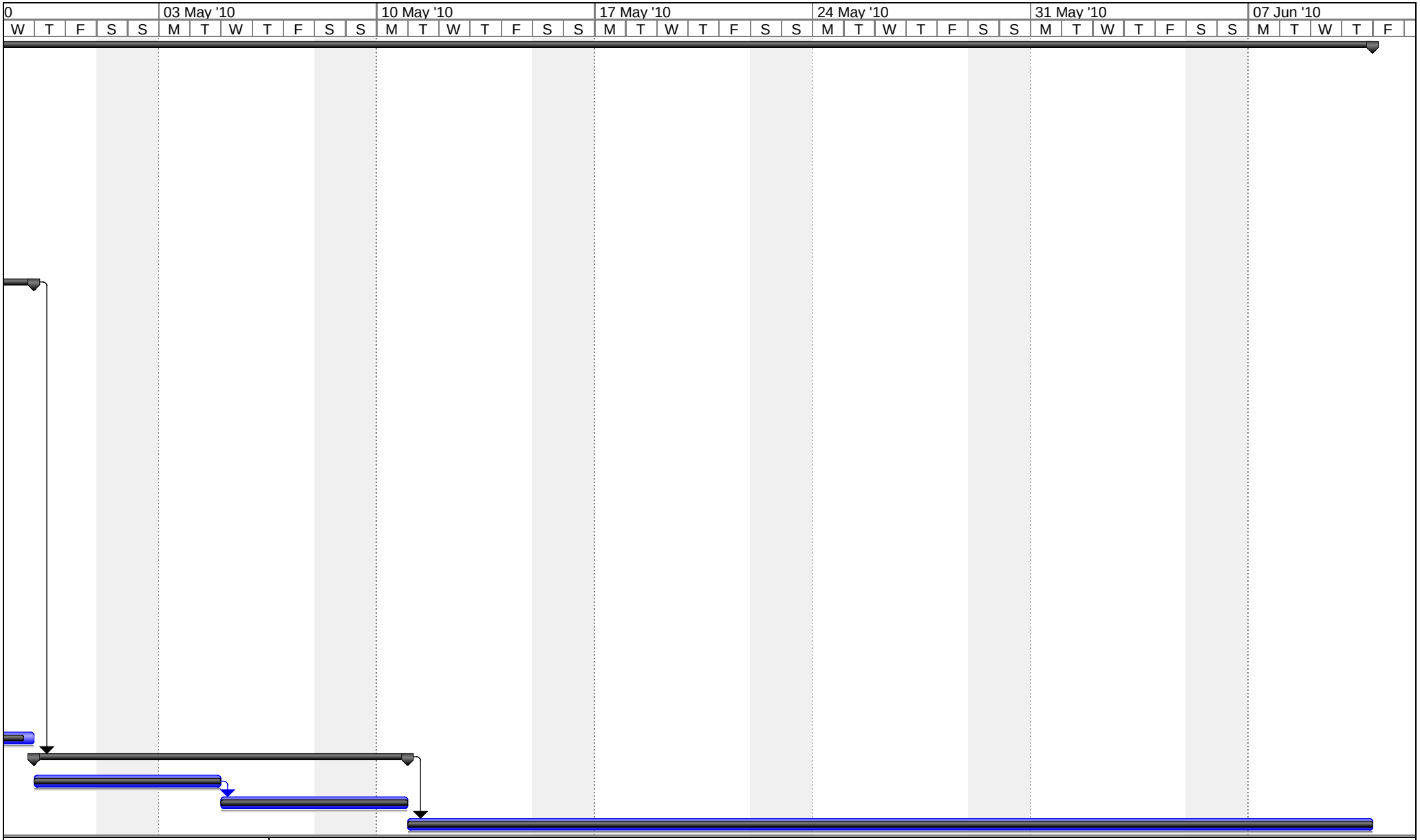
Task		Milestone		External Tasks	
Split		Summary		External Milestone	
Progress		Project Summary		Deadline	


01 Feb '10
08 Feb '10
15 Feb '10
22 Feb '10
01 Mar '10
08 Mar '10
T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S
Project: Implementação de um Servic
Date: Fri 09-07-10
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline



Project: Implementação de um Servic
Date: Fri 09-07-10

Task		Milestone		External Tasks	
Split		Summary		External Milestone	
Progress		Project Summary		Deadline	



Project: Implementação de um Servic
Date: Fri 09-07-10

Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline

ANEXO 2: WSDL base

Este é o WSDL que serviu como base para se perceber que tipo de informação era necessária na identificação de um serviço.

Com este exemplo apenas o serviço HelloWorld funcionaria.

```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/  … >
//---------- Início wsdl:types ----------
   <wsdl:types>
          <s:schema elementFormDefault="qualified" targetNamespace="http:…">
                 <s:element name="HelloWorld">
                        <s:complexType />
                 </s:element>
                 <s:element name="HelloWorldResponse">
                        <s:complexType>
                               <s:sequence>
                                      <s:element    minOccurs="0"    maxOccurs="1"
                               name="HelloWorldResult" type="s:string" />
                               </s:sequence>
                        </s:complexType>
                 </s:element>
                 </s:schema>
   </wsdl:types>
//---------- Fim wsdl:types ----------
//---------- Início wsdl:message ----------
   <wsdl:message name="HelloWorldSoapIn">
          <wsdl:part name="parameters" element="tns:HelloWorld" />
   </wsdl:message>
   <wsdl:message name="HelloWorldSoapOut">
```

```
            <wsdl:part name="parameters" element="tns:HelloWorldResponse" />
    </wsdl:message>
//---------- Fim wsdl:message ----------
//---------- Início wsdl:portType ----------
    <wsdl:portType name="Service1Soap">
            <wsdl:operation name="HelloWorld">
                    <wsdl:input message="tns:HelloWorldSoapIn" />
                    <wsdl:output message="tns:HelloWorldSoapOut" />
                </wsdl:operation>
    </wsdl:portType>
//---------- Fim wsdl:portType ----------
//---------- Início wsdl:binding (soap1) ----------
    <wsdl:binding name="Service1Soap" type="tns:Service1Soap">
            <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
            <wsdl:operation name="HelloWorld">
                    <soap:operation
            soapAction="http://ambisig.pt/Intranet/HelloWorld" style="document" />
                    <wsdl:input>
                            <soap:body use="literal" />
                    </wsdl:input>
                    <wsdl:output>
                            <soap:body use="literal" />
                    </wsdl:output>
            </wsdl:operation>
    </wsdl:binding>
//---------- Fim wsdl:binding (soap1) ----------
//---------- Início wsdl:binding (soap12) ----------
    <wsdl:binding name="Service1Soap12" type="tns:Service1Soap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
            <wsdl:operation name="HelloWorld">
                    <soap12:operation
            soapAction="http://ambisig.pt/Intranet/HelloWorld" style="document" />
                    <wsdl:input>
```

```
                        <soap12:body use="literal" />
                    </wsdl:input>
                    <wsdl:output>
                        <soap12:body use="literal" />
                    </wsdl:output>
                </wsdl:operation>
    </wsdl:binding>
//---------- Fim wsdl:binding (soap12) ----------
//---------- Início wsdl:service ----------
    <wsdl:service name="Service1">
            <wsdl:port name="Service1Soap" binding="tns:Service1Soap">
                    <soap:address location="http://localhost:50456/Service1.asmx" />
            </wsdl:port>
            <wsdl:port name="Service1Soap12" binding="tns:Service1Soap12">
                    <soap12:address  location="http://localhost:50456/Service1.asmx"
            />
            </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
//---------- Fim wsdl:service ----------
```