INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Evolução de Software e Serviços Web

Filipe Emanuel Correia Ramos Santos Moreira
Outubro de 2010

POLITÉCNICO
DO PORTO

Departamento de Engenharia Informática

Evolução de Software e Serviços Web

Filipe Emanuel Correia Ramos Santos Moreira

Tese para obtenção do Grau de Mestre em Engenharia Informática

Área de especialização em Sistemas Gráficos e Multimédias

Orientador: Doutor Alberto A. C. Sampaio

Presidente: Doutor João Paulo Jorge Pereira
Arguente: Doutor Pedro Correia Cravo Pimenta

Porto, Outubro de 2010

Agradecimentos

O desenvolvimento desta Tese é o culminar de um trabalho rigoroso, criterioso e sustentável precedido de uma série de circunstâncias e orientações que tornaram possível a criação deste projecto, o qual, sendo pessoal, não constitui *per se*, um trabalho individual.

O meu primeiro agradecimento, dedico-o aos meus pais. Incansáveis, dedicados e amigos, que sempre me acompanharam, desenhando o meu futuro com as suas possibilidades. Os princípios e os valores que me foram incutidos, tornaram-me na pessoa que sou hoje. O sacrifício compensou. Aos meus Pais, o meu maior agradecimento.

À minha namorada, compreensiva e companheira, dedicada e apaixonada, agradeço a força, perseverança e persistência que me manteve no caminho correcto e orientação nos momentos certos. A ti Vera, muito obrigado.

À minha família, em particular à minha irmã, Dulce, pela paixão, amizade e carinho, à minha tia Marita, pelo apoio, ajuda e aconselhamento, à minha madrinha Albertina, aos meus avós e aos meus primos. Em particular, um abraço especial ao João Pedro, pela amizade e companheirismo nos momentos mais difíceis.

Ao meu orientador, Professor Doutor Alberto Sampaio, um grande obrigado pela disponibilidade, orientação, insistência e atitude positiva. A sua porta sempre aberta, mostrou-me a sua grande capacidade de orientação e liderança, sempre com confiança, qualidade e empenho.

Aos meus amigos, presentes ao longo destes anos no Instituto Superior de Engenharia do Porto, na Universidade de Aveiro, na Auto Sueco, e demais. A todos, obrigado.

O meu último agradecimento vai para o meu grande avô, "Mestre Moreira" que infelizmente já não está presente, mas que sempre esteve e estará presente no meu coração. A si, "Bú", obrigado por tudo.

A todos, um grande obrigado!

Resumo

A Internet causou uma revolução em grande parte dos processos das empresas, criando oportunidades e gerando necessidades até então desconhecidas. Os Sistemas de Informação, ferramentas indispensáveis para uma gestão moderna das actividades empresariais, também foram influenciados pela evolução constante da tecnologia e as facilidades oferecidas para a popularização do uso da Internet. Rumo a uma crescente necessidade de modernização, rapidez, agilidade, eficiência e eficácia que são exigidos das actividades da empresa, actualmente, a tecnologia da Web é exibida como uma opção viável para melhorar o desempenho na gestão da informação.

Esta Tese procura dissertar acerca dos aspectos relativos ao funcionamento e implementação de Serviços Web, suas principais características, vantagens e desvantagens comparativamente aos sistemas de informação tradicionais. Os objectivos propostos podem ser sintetizados no conhecimento e descrição das funcionalidades dos Serviços Web, no estudo da situação actual, na identificação das características que potenciam a capacidade de evolução dos Serviços Web, no estudo e desempenho dos Serviços Web, e finalmente, no estudo e análise da especificação Segurança.

Palavras-Chave: Serviços Web, Redes, Sistemas de Informação, Protocolos, Normas, Internet.

Abstract

The Internet has caused a revolution to a large extent of the enterprise processes, creating chances and generating necessities until then unknown. The Systems of Information, which are indispensable tools for a modern management of the enterprise activities, also have been influenced for the constant evolution of the technology and the facilities offered for the popularization of the use of the Internet. Towards the increasing necessity of modernization, rapidity, agility, effectively and efficiency that are demanded of the enterprise activities currently, the Web technology appears as a viable option to improve the performance in the management of the information.

This Thesis seeks to explain about the aspects concerning the operation and implementation of Web Services, its main features, advantages and disadvantages compared to traditional information systems.

The proposed goals can be synthesized in the knowledge and description of the functionality of Web Services, to study the current situation, to identify the characteristics that enhance the capacity for development of Web Services, the study and performance of Web Services, and finally, the study and analysis of the WS-Security specification.

Keywords: Web Services, Networks, Information Systems, Protocols, Standard, Internet.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos e Abreviaturas

HTML	HiperText Markup Language
HTTP	HyperText Transfer Protocol
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
XML	eXtensible Markup Language
WSDL	Serviços Web Description Language
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
API	Application Programming Interface
ERP	Enterprise Resource Planning
ISP	Internet Service Fornecedor
OASIS	Organization for the Advancement of Structured Information Standards
DMTF	Distributed Management Task Force
SAML	Security Assertion Markup Language

1. Introdução

A criação de Serviços Web tem como objectivo principal o desenvolvimento uma rede interoperável para transporte de dados que pudesse ser usada por todas as empresas e por todos os programadores, independentemente do sistema/ linguagem que estão suportados.

Os Serviços Web evoluem sistematicamente e de forma estruturada. Cada vez mais, o uso de serviços integrados e partilhados é maior. Existe mais e melhor acesso à informação, melhores serviços, com maior qualidade e segurança.

O desenvolvimento de novas interfaces, arquitecturas, integração de novos serviços e dados é premente e urgente. Actualmente, os sistemas de informação são cada vez mais complexos, resultando em sistemas com maiores níveis de dificuldade de compreensão, desenvolvimento e manutenção.

1.1. Objectivos Propostos

Os objectivos propostos para esta Tese, podem ser sintetizados nos seguintes pontos:

- Conhecimento e descrição das funcionalidades dos Serviços Web
- Estudo da situação actual dos Serviços Web
- Identificação das características que potenciam a capacidade de evolução dos Serviços Web
- Estudo e desempenho dos Serviços Web (Performance)
- Estudo e análise da especificação Segurança

Esta Tese procura abordar alguns dos aspectos relativos ao funcionamento de Serviços Web, suas principais características, vantagens e desvantagens em relação aos sistemas de informação tradicionais. Foi elaborada tendo por base uma pesquisa exaustiva, utilizando para tal, algumas das palavras-chave acima descritas. Com base nas pesquisas, foram elaborados

excertos, resumidos, editados e sintetizados. Foram também retirados, excertos originais referenciados pelos seus autores, que tentam completar a ideia a passar.

Foram escolhidos artigos e publicações de referência na área, que, após a compilação da informação recolhida, foi tentada uma sequência lógica e encadeada da informação desenvolvida, para que, ao mesmo tempo, o resultado final deste relatório seja perceptível e satisfatório.

1.2. Metodologia

Definidos os objectivos, pretendo assim, numa primeira instância, fazer uma análise temporal do início do uso dos serviços Web, demonstrar a primeira aplicação do conceito, em que consistiam, para que serviam, como surgiram e com que propósito. Pretendo também analisar os primeiros serviços activos (linguagem, redundância, QoS, disponibilidade, processos e métodos de evolução, tolerância a falhas, …).

Seguidamente, apresentam-se as características, capacidades, potencialidades métodos, normas e processos inerentes ao desenvolvimento, manutenção e assistência dos Serviços Web actuais.

Mantendo um raciocínio coerente, pretendo demonstrar a performance dos Serviços Web através de testes de desempenho e performance com diferentes cargas. Analisando a especificação WS-Security, demonstrarei que a mesma afecta a performance dos Serviços Web.

1.3. Estrutura do documento

Este documento está dividido em 7 secções. A Secção 1 introduz a temática da Tese, apresentando os objectivos propostos, estrutura e notas metodológicas. Na Secção 2 está descrita a origem do conceito, evolução temporal, descrição e características dos Serviços Web. Na Secção 3, são apresentadas e descritas as especificações normativas afectas aos Serviços Web. Na Secção 4, desempenho dos Serviços Web e respectivos testes. Na Secção 5 é analisada a especificação Segurança nos Serviços Web. Posteriormente, no ponto 6, são abordados aspectos como a capacidade evolutiva, futuros desenvolvimentos e conclusões

finais. Finalmente, na Secção 7, a referência bibliográfica consultada para desenvolvimento desta Tese.

2. Serviços Web

Um Serviço Web é um sistema identificado por uma URI, cujas interfaces públicas e ligações são definidas e descritas usando XML. A sua definição pode ser descoberta por outros sistemas. Estes sistemas podem, assim, interagir com o Serviço Web pela sua definição, usando mensagens baseadas em XML transmitidas através de protocolos da Internet [1].

Definição de Serviço Web:

… "um Serviço Web é uma aplicação, acessível na Internet (ou intranet de uma empresa) através de uma URL, que é acedida por clientes através de protocolos baseados em XML, como o *Simple Object Access Protocol* (SOAP), enviada por meio de protocolos de Internet, como o HTTP. Os clientes acedem a uma aplicação (Serviço Web) através das suas interfaces e ligações, que são definidas por meio de XML, tais como ficheiros WSDL (*Web Services Definition Language*) [33].

Nos dias de hoje, uma das mais importantes tendências no campo das Tecnologias de Informação é o impulso para uma integração de dados, simplificando sistemas, utilizando funcionalidades e reduzindo redundâncias. Durante anos as empresas têm tentado integrar dados através de vários métodos, mas hoje existem dois que se destacam dos restantes: EAI (*Enterprise Application Integration*) e Serviços Web. Embora estes dois métodos sejam profundamente considerados como inovações distintas, a abordagem para o problema da desarticulação de dados empresariais é um exemplo de como as ideias em tecnologias da informação evoluem continuamente com a ocorrência de ambas.

A EAI é, por definição, a tradução de dados e comandos de um formato de uma aplicação para o formato de outra. É uma comunicação contínua entre dois ou mais sistemas anteriormente incompatíveis tornada possível através de redes com o propósito de permitir a partilha de dados empresariais e a racionalização das funcionalidades. Do mesmo modo, Serviços Web são definidos como aplicações baseadas na Web, definidas em padrões de interacção díspares entre as aplicações que permitem também às empresas partilhar os dados e invocar processos entre diferentes aplicações [8].

As semelhanças entre estas duas tecnologias existem devido à sua origem comum em interfaces de programação de aplicações (API's) e componentes reutilizáveis, que os profissionais das Tecnologias de Informação têm vindo a utilizar durante anos para resolver o mesmo tipo de problemas de integração.

Inicialmente, foram utilizadas API's para expor sistemas para outro formato de dados e processos que poderiam ser partilhados entre aplicações de uma empresa. Esta abordagem foi intrinsecamente imperfeita, porque para uma aplicação chamar uma API, esta tinha de estar numa plataforma compatível. Isto significava que os sistemas construídos em plataformas diferentes, tecnologias ou linguagens de programação só poderiam integrar, se uma API fosse construída na tecnologia de cada aplicação necessária para utilizá-la (ou seja, CICS, COM e API's Java para o mesmo conjunto aplicacional). Este método revelou-se pesado e difícil de manter, razão pela qual não está actualmente a ser utilizado.

As API's partem do princípio de serem utilizadas para expor sistemas e integrar dados, aliado à evolução de componentes reutilizáveis (também designado por objectos), e serem construídas e desenvolvidas sobre um princípio simples – representar entidades comuns que podem ser utilizadas noutras aplicações. Esta componente geralmente representa dados de um cliente provenientes de uma base de dados comum. O resultado ideal é a utilização de um controlo de dados dos clientes ao longo de todas as aplicações numa organização. Analisando especificamente o efeito dos objectos – a tentativa de integrar a informação a partir de vários pontos dentro de uma empresa – torna-se evidente que esta evolução está relacionada com a EAI e Serviços Web.

Embora o conceito original de componentes reutilizáveis tenha sido positivo em termos de integração e gestão de dados, a execução não foi tão bem sucedida. Os componentes reutilizáveis não permitem facilmente a integração de plataformas e tecnologias diferentes, porque não há uma interface universal em que todas as tecnologias possam aceder. Uma empresa que adopte uma arquitectura de componentes reutilizáveis, terá que reescrever todas as aplicações numa plataforma ou tecnologia, ou produzir API's para os componentes em múltiplas linguagens de programação. Isto é pouco prático e relativamente pesado.

Nos últimos anos, EAI têm evoluído de interfaces de dados para pacotes de software. Como a arquitectura de ligação Java (JCA) tornou-se parte da especificação Java, foi utilizada por fornecedores com base em normas para criar interfaces para aplicações.

À medida que a Internet se tornou mais sofisticada com linguagens de programação e protocolos normalizados, como o XML, *Simple Object Access Protocol* (SOAP), e HTTP (*Hypertext Transfer Protocol*) a integração das aplicações evoluiu para a próxima geração de Serviços Web.

O conceito de Serviços Web consiste em integrar diferentes sistemas independentemente da plataforma ou tecnologia, utilizando a Internet e as tecnologias Web. No intuito de criar este cenário, uma página web é disponibilizada para que um sistema possa correr comandos SOAP. SOAP tornou-se, assim, um protocolo comum sobre a qual estas aplicações podem interagir e partilhar informações.

Os Serviços Web fornecem uma arquitectura flexível para a construção de sistemas distribuídos com interoperabilidade universal. Utilizam mensagens XML para o pacote de dados definido pelo SOAP (*Simple Object Access Protocol*) para descrever os tipos de dados e serviços. Com os Serviços Web, várias aplicações de diferentes organizações podem ser facilmente integradas, mesmo sendo desenvolvidas em diferentes linguagens de programação e implementadas em diferentes plataformas. Desta forma, os Serviços Web têm sido profundamente adoptados no mercado como uma plataforma padrão de tecnologia *middleware* independente.

Em síntese, Serviços Web têm a capacidade de colocar duas aplicações a comunicar através de um padrão de dados e comandos de representação semântica (XML e SOAP). Em vez de se basear na plataforma ou tecnologia subjacente de cada aplicação, os padrões criam os meios para essa comunicação.

Com esta interacção baseada na Web, é agora possível, por exemplo, em tempo real, uma aplicação Cobol correr numa *mainframe* para integrar com uma aplicação Java que corre num servidor UNIX, sem as questões que existiam na integração com os componentes reutilizáveis e métodos de adaptação EAI. Neste ponto, os Serviços Web são os mais sofisticados meios de integração, seja dentro de uma organização, seja na Web.

O futuro da integração reside na aplicação de Serviços Web que se transformaram e evoluíram, tal como os componentes reutilizáveis e adaptadores EAI. O ponto mais importante é perceber que todas estas tecnologias e métodos foram desenvolvidos para o mesmo objectivo: permitir que diferentes aplicações comuniquem umas com as outras, partilhem dados e permitam alavancar novas funcionalidades [6, 7, 8].

2.1. Como Surgiram

Como é profundamente conhecido, EDI (*Electronic Data Interchange*), lançado em 1975, foi a primeira tentativa de criar um formato padrão para que as empresas pudessem comunicar entre si através de uma rede. EDI é um formato padrão de troca de dados. O padrão é ANSI X12 e foi desenvolvido pela *Data Interchange Standards Association*. ANSI X12 está estreitamente coordenada com a norma internacional EDIFACT e está a ser fundida na mesma.

Uma mensagem EDI contém uma *string* de elementos de dados, cada uma das quais representa um registo singular, tais como um preço, o ID, etc, separados por delimitadores. Uma ou mais *strings* de dados é constituída por um cabeçalho e uma operação, formando uma transacção, que é a unidade de transmissão EDI (equivalente a uma mensagem). Mensagens EDI são um meio de comércio electrónico, que abrange também o e-mail, fax e, podem ser encriptadas.

Desde que EDI surgiu, existiram numerosas tentativas de uma conduta universal para a ligação lógica empresarial através de uma rede: *Common Object Request Broker Architecture*, *Distributed Component Object Model*, Chamada de Procedimento Remoto Unix, e Método Invocação Remoto Java. Nenhuma destas tecnologias conseguiu ganhar mercado significativo ou impulso suficiente para alcançar o sucesso. Todas elas existem hoje em dia.

A Web usa o protocolo HTTP que corre sobre TCP/ IP. TCP/ IP já era um padrão consistente no momento em que a Web se tornou numa *mainstream* em 1994, e, até 1997, o HTTP tinha-se tornado um padrão universal.

Foi a criação do XML que realmente abriu o caminho para os Serviços Web, pois sendo uma notação padrão independente para a descrição de dados, o XML foi uma escolha lógica para o trabalho de padronização de comunicação "ponto-a-ponto", utilizado para descrever protocolos de passagem de mensagens.

XML tornou-se um padrão oficial em Fevereiro de 1998, quando a *World Wide Web Consortium* anunciou que chegada da versão 1.0 do XML, recomendada para a implementação em aplicações.

Até início de 1998 foram feitas várias tentativas de criar um protocolo XML para comunicação entre processos. O *Web Distributed Data Exchange* (WDDX) da empresa

ALLAIRE Corp foi uma tentativa independente, mas foi o SOAP, desenvolvido por Dave Winer, CEO da Userland Software, Bob Atkinson e Mohsen Al-Ghosein, engenheiros da Microsoft, e Don Box, co-fundador da DevelopMentor Incorporated, que viria a ser a base para Serviços Web [5].

Os mercados electrónicos foram um conceito activo em Dezembro de 1999, quando a Microsoft realizou uma reunião privada com a IBM e outras empresas interessadas no SOAP 1.0, a especificação de um protocolo padronizado para troca de mensagens baseadas em XML.

A IBM foi a primeira a apostar em SOAP. Em Março de 2000, a IBM apoiada publicamente pela Microsoft começou a trabalhar no desenvolvimento do SOAP 1.1.

Até ao Verão de 2000, SOAP foi ganhando uma maior aceitação. IBM e Microsoft foram também trabalhando numa forma de descrever a ligação a um Serviço Web, em termos de programação [6].

Após alguma discussão, as propostas de protocolo entre a Microsoft e IBM fundiram-se. A IBM contribuiu com *Network Accessible Service Specification Language* e a Microsoft com *Service Description Language* e SOAP. No Outono de 2000, a especificação, *Web Service Description Language* foi anunciada.

Com o SOAP e a WSDL, as empresas poderiam criar e qualificar os seus Serviços Web. Mas ainda era necessário fornecer uma forma de anunciar e localizar Serviços Web. Em Março de 2000, IBM, Microsoft e Ariba começaram a trabalhar na solução: "*Universal Description, Discovery and Integration*". Em Setembro de 2000, foi anunciado UDDI 1.0.

Com o SOAP, WSDL e UDDI em vigor, as normas para criar Serviços Web tinham chegado, mas foi até o final de 2000 que cinco grandes empresas de TI (Oracle, HP, Sun, IBM e Microsoft) anunciaram o seu compromisso para com os Serviços Web declarando a sua intenção de apoiar e implementar Serviços Web nos seus produtos.

O aparecimento dos Serviços Web no mercado em 2002 contribuiu para a evolução do serviço *Business-to-Consumer* (B2C) uma área em expansão, com Serviços Web orientados ao consumidor.

Em 2003, ocorre a adopção de Registos UDDI. O ano 2004 conhece a maturidade dos Serviços Web baseados em modelos de negócio, onde se regista um aumento de 40% nas transacções de serviços financeiros e 35% nos serviços governamentais *online*.

De 2005 em diante, registou-se uma mudança comportamental ao nível das empresas, as organizações alteraram os seus processos de negócio, e os seus modelos de negócios que se orientaram tendencialmente para a colaboração em tempo real e interoperabilidade, quer dentro das empresas, quer entre empresas.

Graças ao desenvolvimento na área das tecnologias de informação e comunicação nos últimos anos, várias pessoas e empresas possuem agora melhores e mais rápidas ligações à Web, usando-a de forma proactiva. Através do uso de *browsers*, diferentes plataformas e aplicações Web puderam, finalmente, interagir entre si [5, 6, 7].

2.2. Situação Actual

Os Serviços Web surgiram para o desenvolvimento de aplicações de negócio, suportando a colaboração e a negociação de forma aberta e transparente, distribuída e dinâmica entre entidades. Existe uma forte tentativa de normalização das tecnologias envolvidas.

Um Serviço Web é qualquer software que está disponível na Internet através de uma interface XML. XML é utilizado para codificar toda a comunicação de/ para um Serviço Web. Estas são as peças essenciais nos Serviços Web para se construir aplicações distribuídas.

Usando Serviços Web, qualquer aplicação pode publicar as suas funções ou mensagens para os restantes utilizadores da Web.

Os motivos para usar Serviços Web prendem-se com a utilização de normas, maior facilidade de desenvolvimento, têm tipos pré-definidos (string, int, long, array, etc) que funcionam com Java, .Net, C/ C++, etc, são independentes da linguagem e do sistema operativo, funcionam com HTTP ou SMTP, e normalmente não tem problemas com *firewalls*.

As tecnologias subjacentes aos Serviços Web (tais como HTTP, SOAP, WSDL, UDDI, XML) são abertas, largamente difundidas e consensuais. No entanto, existem críticas que demonstram receios ou falsas expectativas em que os investimentos em Serviços Web podem suscitar. Uma crítica construtiva diz respeito ao facto do SOAP ser menos eficiente do que os

sistemas RPC existentes (as mensagens trocadas são descritas em formato XML enquanto nos sistemas RPC são trocadas em formato binário).

No entanto, esta desvantagem é contrabalançada pela facilidade de interoperabilidade entre os serviços, sem os problemas conhecidos de segurança e/ ou *firewalls*.

O termo Serviço Web descreve um padrão de integração de aplicações baseadas na Web utilizando padrões XML, SOAP, WSDL e UDDI através da Internet.

A definição de Web Service do W3C abrangem diferentes sistemas, mas no seu sentido mais lato, o termo refere-se a clientes e servidores que comunicam entre si através do protocolo HTTP usado na Web.

Os Serviços Web são construídos com várias tecnologias que trabalham em articulação com os padrões emergentes para garantir a segurança e gestão das aplicações e serviços.

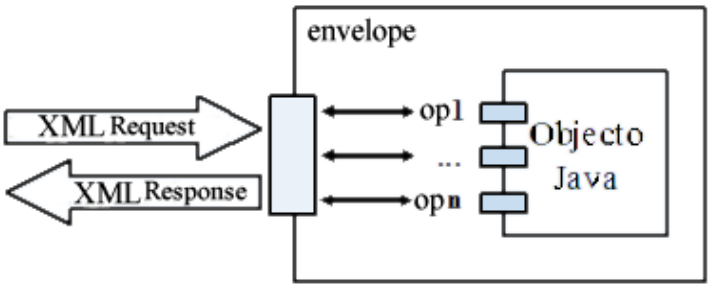

Figura 1 - Exemplo do fluxo de informação de um Serviço Web [8]

"A few years ago Web Services were not fast enough to be interesting."

- www.w3schools.com –

VANTAGENS	**DESVANTAGENS**
Standard W3C	Performance
Comunicação entre sistemas heterogéneos	Não é transaccional
Standard para comunicação entre	A interface não deve mudar sem avisar os

plataformas	utilizadores
Interoperabilidade	
Disponibilização de serviços	
Integração com sistemas	
Liberdade de escolha	
Suporte a vários tipos de cliente	
Aumento de produtividade	

Tabela 1 - Vantagens e desvantagens dos Serviços Web

Os Serviços Web podem ser divididos em duas áreas: "*Big Web Services*" e "*Restful Web Services*".

"*Big Web Service"* utilizam mensagens XML que utilizam o padrão SOAP e são populares entre empresas tradicionais. Nestes sistemas, existem descrições dos serviços e operações oferecidas pelo serviço através de *Web Service Description Language* (WSDL). Este último não é uma exigência de SOAP, mas é uma condição necessária do lado do cliente.

Mais recentemente, "*Restful Web Services*" têm recuperado popularidade. Estes também se revêem especificados no W3C, e, frequentemente, permitem melhores integrações HTTP do que serviços baseados em SOAP. Não exigem WSDL nem definições do serviço.

2.3. Protocolos dos Serviços Web SOAP

Os Serviços Web são utilizados principalmente como um meio para as empresas comunicarem entre si e com clientes, permitindo comunicar dados entre organizações sem profundo conhecimento mútuo dos sistemas de informação por trás da firewall.

Como já foi referido, uma pilha de protocolos são usados para definir, localizar, implementar e permitir que os Serviços Web interajam entre si: SOAP, WSDL, UDDI, XML.

O XML é usado para codificar os dados, enquanto que o SOAP é usado para transferir dados. WSDL é usado para descrever os serviços disponíveis e UDDI é usado para listar os serviços que estão disponíveis.

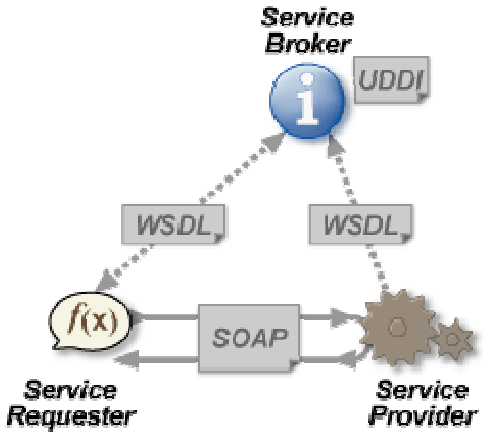

Figura 2 – Arquitectura de Serviços Web [34]

2.3.1. SOAP

O SOAP (*Simple Object Access Protocol*) é um protocolo de comunicação baseado em mensagens XML para aceder a um Serviço Web. É usado para codificar a informação na Web na solicitação de mensagens e respostas antes de enviá-las através da rede. As mensagens SOAP são independentes de qualquer sistema operativo ou protocolo e podem ser transportadas usando uma variedade de protocolos da Internet, incluindo SMTP, MIME, HTTP, permite evitar as *firewalls* e é um padrão W3C [1].

Soap Protocol Stack

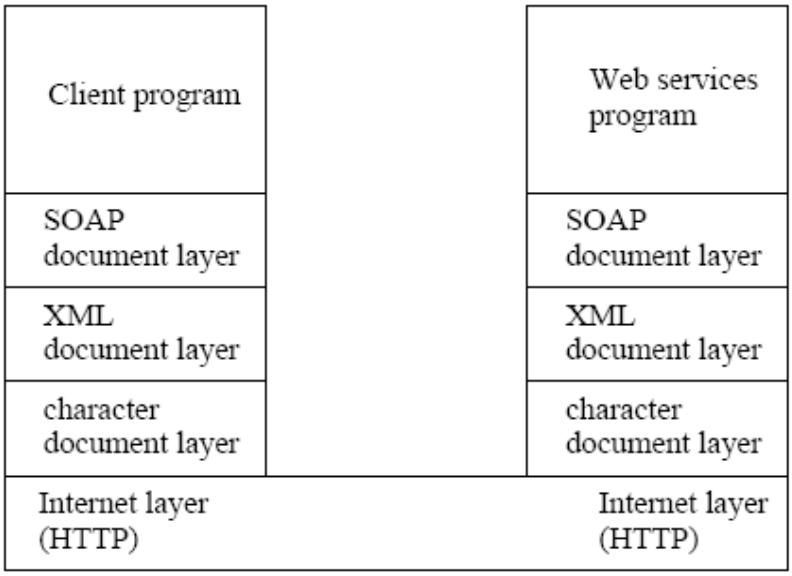

Figura 3 – Soap Protocol Stack [2]

Mensagem SOAP

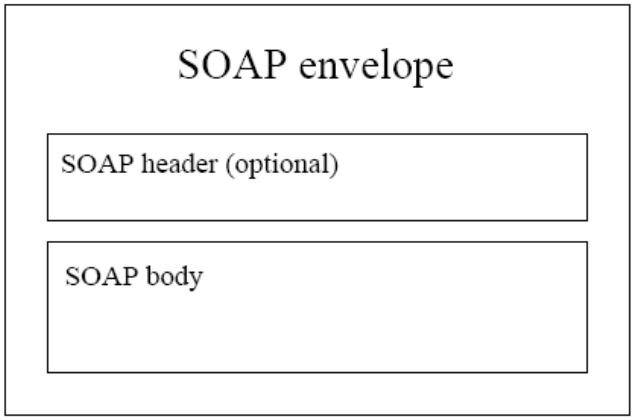

Figura 4 - Estrutura de uma mensagem SOAP [1]

Exemplo de um pedido SOAP

No exemplo de código abaixo, está patente uma invocação de um serviço SOAP, em que é solicitada a temperatura de uma localidade, neste caso concreto, do Porto.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      <SOAP-ENV:Body>
            <ns1:getWeatherxmlns:ns1="urn:examples:weatherservice"
            SOAPENV:encodingStyle="http://www.w3.org/2001/09/so
            <localidade xsi:type="xsd:string">Porto</localidade>
            </ns1:getWeather>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo de uma resposta SOAP

No exemplo de código abaixo, está patente uma resposta de um serviço SOAP, em que é retornada a temperatura de uma localidade, neste caso concreto, 24.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      <SOAP-ENV:Body>
            <ns1:getWeatherResponse
            xmlns:ns1="urn:examples:weatherservice"
            SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/so
            <return xsi:type="xsd:int">24</return>
            </ns1:getWeatherResponse>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.3.2. WSDL

A WSDL (*Web Service Description Language*) é uma linguagem de formato XML usado para descrever as capacidades de um Serviço Web como conjuntos de serviços capazes de trocar mensagens. WSDL é fundamental para o UDDI, que se descreve abaixo.

WSDL foi desenvolvida em conjunto pela Microsoft e a IBM e é um padrão W3C.

A WSDL permite descrever os Serviços Web e como podem ser acedidos. Para tal, especifica todos os métodos (e parâmetros) disponibilizados aos possíveis programas cliente.

A interface inclui:

- o nome dos métodos que o Serviço Web aceita.
- os parâmetros que os métodos aceitam.
- o formato da respostas que devolve.
- o protocolo de transporte utilizado (tipicamente HTTP).
- quais os URL's para o serviço.

2.3.3. UDDI

O UDDI (*Universal Description, Discovery and Integration*) é um directório distribuído baseado na Web onde as empresas podem registar os seus Serviços Web e procurar outros, sendo portanto, um directório para armazenar informação acerca dos Serviços Web. Os Serviços Web ser descritos em WSDL. O UDDI comunica via SOAP.

2.3.4. XML

XML permite o uso de uma linguagem que pode ser usada por diferentes plataformas e diferentes linguagens de programação. Permite, ainda, a troca de mensagens complexas e funções.

Sigla de *eXtensible Markup Language*, uma norma desenvolvida pela W3C, o XML tem por base a notação SGML, projectada especialmente para documentos da Web. Permite criar as suas próprias etiquetas, permitindo a definição, transmissão, validação e interpretação de dados entre aplicações e entre organizações.

2.4. REpresentational State Transfer (REST)

Actualmente existem duas escolas de pensamento no desenvolvimento de Serviços Web: a abordagem tradicional, baseada em padrões (SOAP) e a conceptualmente mais simples e mais recente, *REpresentational State Transfer* (REST).

Na engenharia de software, o termo *"software architectural style"* refere-se geralmente a "um conjunto de regras de concepção que identificam os tipos de componentes e conectores que podem ser utilizados para compor um sistema ou subsistema."

Alguns exemplos comuns de *"software architectural style"* incluem *"Pipe and Filter", "Layered", "Push Based"*.

No mundo dos Serviços Web, *REpresentational State Transfer* (REST) é uma chave que conjuga uma arquitectura cliente-servidor em que os Serviços Web são vistos como recursos e podem ser identificadas pelas suas URL's.

Clientes dos Serviços Web que pretendam utilizar estes recursos de acesso a uma representação particular, transferem conteúdos usando um pequeno conjunto de métodos remotos definidos globalmente que descrevem a acção a ser realizada sobre o recurso.

REST é uma descrição analítica da actual arquitectura Web, e assim, a interacção entre o estilo e o protocolo HTTP resulta sem falhas.

Os métodos HTTP, GET e POST são os verbos que o programador usa para descrever as acções necessárias, Criar (Create), Ler (Read), Actualizar (Update) e Eliminar (Delete) (CRUD).

É possível ver uma analogia às operações SQL, que também depende de alguns verbos comuns, conforme demonstrado na Tabela 2. No entanto, REST e HTTP são mutuamente exclusivos, e, no entanto, REST não requer HTTP.

Acção	SQL	http
Create	`Insert`	`PUT`
Read	`Select`	`GET`

Update	`Update`	`POST`
Delete	`Delete`	`DELETE`

Tabela 2 - Relação entre métodos SQL e HTTP [11]

Suporte RESTful em JAX-WS

A API Java para Serviços Web XML, designada por JAX-WS, fornece todo o apoio para a construção e implementação de Serviços Web RESTful. A API foi desenvolvida através do programa *Java Community Process* como JSR 224. É perfeitamente integrada com *Java Architecture for XML Binding* (JAXB) para vincular dados XML com a tecnologia Java e está incluída em *Java Platform*, *Standard Edition* (Java SE) 6 e *Java Platform, Enterprise Edition* (Java EE) 5.

2.5. REST versus SOAP

Os Serviços Web SOAP e REST têm filosofias muito diferentes. SOAP é realmente um protocolo baseado em XML para a computação distribuída, enquanto que REST prefere ambientes virtualizados baseados na Web [11, 32].

São os profissionais que devem decidir que estilo é apropriado para as suas aplicações. Um desenvolvimento RESTful pode ser apropriado quando:

- Os Serviços Web são completamente "*stateless*". Um bom teste é determinar se a interacção pode sobreviver a um "*restart*" do servidor.
- Uma infra-estrutura em cache pode ser aproveitada para o desempenho. Se os dados que o Serviço Web devolve não são gerados dinamicamente e podem ser armazenados então, o armazenamento em cache que os servidores Web e outros intermediários intrinsecamente fornecem, podem ser aproveitados para melhorar o desempenho. No entanto, o programador deve ter cuidado, porque estas caches estão limitadas ao método HTTP GET para a maioria dos servidores.
- O serviço produtor e o serviço consumidor (*service producer and service consumer*) tenham um entendimento mútuo sobre o contexto e o conteúdo a ser transmitido. Uma vez que não existe uma forma padrão para descrever as interfaces dos Serviços

Web, ambos devem concordar com os "*Schemas*" que descrevem os dados trocados e as formas de processamento.

- A largura de banda é particularmente importante e deve ser limitada. REST é particularmente útil para dispositivos como PDA's e telemóveis, para que a sobrecarga de cabeçalhos e camadas adicionais de elementos SOAP e XML seja limitada.

- Disponibilização de Serviços Web ou agregação em sites pode ser facilmente activado com um estilo RESTful. Os programadores podem utilizar tecnologias como *Asynchronous JavaScript* com XML (AJAX) e ferramentas tais como *Direct Web Remoting* (DWR) para consumir os serviços nas suas aplicações Web. Os serviços podem ser expostos com XML e consumidos através de HTML sem refazer a arquitectura do actual site.

Desenvolvimento baseado em SOAP pode ser adequado quando:

- Um contrato formal é estabelecido para descrever a interface que oferece o Serviço Web. *Web Services Description Language* (WSDL) descreve os detalhes, tais como mensagens, operações, ligações, e localização dos Serviços Web.

- A arquitectura deve assegurar requisitos complexos e não funcionais. Muitas especificações dos Serviços Web garantem essas exigências e estabelecem um vocabulário comum entre eles. Exemplos como *Transactions, Security, Addressing, Trust, Coordination.* A maioria das aplicações transcendem-se e evoluem para além das simples operações CRUD e exigem informações contextuais e de conversação do estado a ser mantido. Com a abordagem RESTful, os programadores deverão transpor estas questões para a camada de aplicação.

- A arquitectura tem de lidar com processamento assíncrono e invocação. Nesses casos, a infra-estrutura fornecida por padrões, como WSRM e API's, como JAX-WS do lado do cliente apoiada na invocação assíncrona pode ser aproveitada.

	SOAP	REST
Vantagens:	• Linguagem, plataforma, e transporte agnósticos • Desenvolvido para permitir ambientes de computação distribuída • Prevalece como padrão e permite melhor suporte (WSDL, WS-*) • Melhor tolerância a falhas • Escalabilidade	• Linguagem e plataforma agnósticas • Mais simples de desenvolver • Menor curva de aprendizagem • Ferramentas mais intuitivas • Concisa, não necessita da camada de Mensagens • Filosofia e design baseados na Web
Desvantagens:	• Conceptualmente mais complicado, e mais pesado • Muitos métodos • Desenvolvimento mais difícil e menos intuitivo • Requer ferramentas	• Assume um modelo de comunicação ponto-a-ponto • Falta de padrões e normas de suporte para segurança, mensagens confiáveis, etc. • "Preso" ao transporte HTTP

Tabela 3 – Vantagens e desvantagens de SOAP vs REST [11]

Tendo em conta a maior simplicidade da arquitectura REST para o desenvolvimento de Serviços Web, no próximo subcapítulo será ilustrado e analisado um exemplo de implementação de um Serviço Web típico de negócio mostrando os cenários do lado do servidor (recepção, processamento e resposta de solicitações) e do lado do cliente (envio de solicitações).

2.6. Exemplo práctico de um Serviço Web usando REST

O exemplo que vai ser analisado é um *endpoint* que originalmente continha um único método - acceptPO - que aceita uma ordem de compra (*PurchaseOrder)* e retorna um estado (*PurchaseOrderStatus)* [11].

A Tabela 4 mostra mais algumas operações CRUD a adicionar a este serviço.

Descrição	Método Java
Criar um novo Pedido de Compra	`public Purchase OrderStatus acceptPO(PurchaseOrder order)`

Obter dados de Pedido de Compra existente	`public PurchaseOrder retrievePO (String orderID)`
Modificar um Pedido de Compra existente	`public PurchaseOrder updatePO(PurchaseOrder order)`
Cancelar Pedido de Compra	`public void cancelPO(String orderID)`

Tabela 4 - Operações CRUD Adicionais [11]

JAX-WS permite construir RESTful *endpoints* através de uma interface `javax.xml.ws.Provider` na API.

Cada ordem de compra é submetida a um fornecedor, pelo que é usada uma interface genérica `Provider` que pode ser implementada por uma classe como uma alternativa dinâmica para um *service endpoint interface* (SEI). Um serviço implementado através desta interface pode ser suportado num Java EE *container* ou publicado em modo *stand-alone* através da API JAX-WS.

O `Provider` contém um método único com a seguinte assinatura:

```
T invoke(T request)
```

`Provider` é uma API de baixo nível, mas usando-o, implica que o *endpoint* tenha um conhecimento profundo da mensagem pretendida ou da estrutura da carga a ser passada para o serviço. Dependendo de como o `Provider` for implementado, os tipos suportados para T e as suas utilizações são as seguintes:

- `Javax.xml.transform.Source.` Permite que o `Provider` produza e consuma o XML directamente;
- `Javax.activation.DataSource.` Trabalha com mensagens MIME;
- `Javax.xml.soap.SOAPMessage.` Trabalha e manipula toda a mensagem SOAP;

A anotação `ServiceMode` é utilizada para configurar o modo de mensagens de uma instância `Provider`. Com `@ServiceMode(value=MESSAGE)`, o fornecedor irá receber e devolver todo protocolo de mensagens.

Com `@ServiceMode(value=PAYLOAD)`, o *runtime* vai apenas passar o *payload* da mensagem para o fornecedor. Isto é útil quando se quer construir um WSDL baseado em Serviços Web, mas se pretende aceder directamente ao XML e retornar também XML.

O exemplo de código 1 [11] mostra um extracto da classe `PurchaseOrderService` que implementa a interface `Provider`. O serviço processa os quatro principais métodos HTTP e invoca as operações de negócio no seu contexto. A Tabela 2 apresenta a operação invocada, a amostra HTTP solicitada, a resposta HTTP, e o método.

Exemplo de Código 1 (Servidor com operações CRUD)

```
@javax.xml.ws.WebServiceProvider
@javax.xml.ws.ServiceMode(value=javax.xml.ws.Service.Mode.MESSAGE)

//Definição da Classe Pedido de compra
public class PurchaseOrderService implements Provider<Source>
{
 private JAXBContext jc;
 @javax.annotation.Resource(type=Object.class)
 protected WebServiceContext wsContext;

//Método definição do contexto do serviço
 public PurchaseOrderService()
 {
  try {
   jc = JAXBContext.newInstance("com.sun.examples.rest");
  } catch(JAXBException je) {
    throw new WebServiceException("Cannot create JAXBContext", je);
  }
 }

//Método tratamento do serviço solicitado
public Source invoke(Source source)
{
 try {
  MessageContext mc = wsContext.getMessageContext();
  String path = (String)mc.get(MessageContext.PATH_INFO);
  String method = (String)mc.get(MessageContext.HTTP_REQUEST_METHOD);
```

```
  if (method.equals("GET"))
   return get(mc);
  if (method.equals("POST"))
   return post(source, mc);
  if (method.equals("PUT"))
   return put(source, mc);
  if (method.equals("DELETE"))
   return delete(source, mc);

  throw new WebServiceException("Unsupported method:" +method);
 } catch(JAXBException je) {
  throw new WebServiceException(je);
 }
}
```

Com Serviços Web RESTful, existe um mapeamento natural entre os métodos HTTP e as operações CRUD que muitos serviços expõem. Embora não exista uma regra normalizada, as seguintes orientações são aplicáveis na maioria dos casos:

- GET é usado para obter os dados ou realizar uma pesquisa sobre um recurso. Os dados devolvidos a partir do Serviço Web são uma representação do recurso.
- POST é usado para criar um novo recurso. O Serviço Web pode responder com dados ou *status* indicando sucesso ou fracasso.
- PUT é usada para actualizar os recursos existentes ou dados.
- DELETE é utilizado para remover um recurso ou dados.

Em alguns casos, as acções de actualização e eliminação podem ser realizadas com operações POST, por exemplo, quando os serviços consumidos por *browsers* que não suportam PUT ou DELETE. O *GlassFish Application Server* e a API JAX-WS suportam todas as quatro operações HTTP demonstradas na Tabela 2.

A Tabela 5 mostra as mensagens de solicitação HTTP e resposta HTTP para a operação de execução do serviço de encomenda.

Operação	HTTP Request	HTTP Response	Java Technology Method
Create	POST /restfulwebservice-war/poservice/ HTTP/1.0 Accept: */* Connection: close Content-Type: text/xml Content-Length: 618 Pragma: no-cache <tns:PurchaseOrderDocument xmlns:tns="urn:PurchaseOrderDocument"> <billTo> <street>1 Main Street</street> <city>Beverly Hills</city> <state>CA</state> <zipCode>90210</zipCode> </billTo> <createDate>2004-03-27T12:21:02.055-05:00</createDate> <poID>ABC-CO-19282</poID> <items> <itemname>Copier Paper</itemname> <price>10</price> <quantity>2</quantity> </items> <items> <itemname>Toner</itemname> <price>920</price> <quantity>1</quantity> </items> <shipTo> <street>1 Main Street</street> <city>Beverly Hills</city> <state>CA</state> <zipCode>90210</zipCode> </shipTo> </tns:PurchaseOrderDocument>	HTTP/1.1 200 OK X-Powered-By: Servlet/2.5 Content-Type: text/xml Date: Fri, 21 Jul 2006 17:07:15 GMT Connection: close <?xml version="1.0" encoding="UTF-8"?> <ns2:Status xmlns:ns2="urn:Status" xmlns:ns3="urn:PurchaseOrderDocument" xmlns:ns4="urn:POProcessingFault"><orderid>ABC1153501634787</orderid> <timestamp>Fri Jul 21 13:07:14 EDT 2006</timestamp></ns2:Status>	public PurchaseOrderStatus acceptPO(PurchaseOrder order)
Read	GET /restfulwebservice-war/poservice/ABC1153501634787 HTTP/1.0 Connection: close Content-Type: text/xml	HTTP/1.1 200 OK X-Powered-By: Servlet/2.5 Content-Type: text/xml Connection: close <?xml version="1.0" encoding="UTF-8"?><ns3:PurchaseOrderDocument xmlns:ns3="urn:PurchaseOrderDocument" xmlns:ns2="urn:Status" xmlns:ns4="urn:POProcessingFault"><billTo><street>1 Main Street</street> <city>Beverly Hills</city><state>CA</state	public PurchaseOrder retreivePO (String orderID)

		><zipCode>90210</zipCode></billTo> <createDate>2006-07-21T13:08:37.505-04:00</createDate> <items><itemname>Copier Paper</itemname><price>10</price><quantity>2</quantity></items> <items><itemname>Toner</itemname><price>920</price><quantity>1</quantity></items> <poID>/ABC1153501634787</poID><shipTo><street>1 Main Street</street><city>Beverly Hills</city><state>CA</state><zipCode>90210</zipCode></shipTo></ns3:PurchaseOrderDocument>	
	GET /restfulwebservice-war/poservice/ HTTP/1.1 Connection: keep-alive	HTTP/1.1 400 Bad Request X-Powered-By: Servlet/2.5 Content-Type: text/xml <?xml version="1.0" encoding="UTF-8"?><ns4:POProcessingFault xmlns:ns4="urn:POProcessingFault" xmlns:ns2="urn:Status" xmlns:ns3="urn:PurchaseOrderDocument"> <message>Unable to retrieve the order associated with the orderid you specified</message> </ns4:POProcessingFault>	Indicates a problem finding the order
Update	PUT /restfulwebservice-war/poservice/ HTTP/1.0 Connection: close Content-Type: text/xml Content-Length: 620 Pragma: no-cache <tns:PurchaseOrderDocument xmlns:tns="urn:PurchaseOrderDocument"> <billTo> <street>1 Main Street</street> <city>Beverly Hills</city> <state>CA</state> <zipCode>90210</zipCode> </billTo> <createDate>2004-03-27T12:21:02.055-05:00</createDate> <poID>ABC-CO-19282</poID> <items> <itemname>Copier Paper</itemname> <price>10</price> <quantity>2</quantity> </items>	HTTP/1.1 200 OK X-Powered-By: Servlet/2.5 Content-Type: text/xml <?xml version="1.0" encoding="UTF-8"?><ns3:PurchaseOrderDocument xmlns:ns3="urn:PurchaseOrderDocument" xmlns:ns2="urn:Status" xmlns:ns4="urn:POProcessingFault"> <billTo><street>1 Main Street</street><city>Beverly Hills</city><state>CA</state><zipCode>90210</zipCode></billTo> <createDate>2004-03-27T12:21:02.055-05:00</createDate> <items><itemname>Copier Paper</itemname><price>10</price><quantity>2</quantity></items> <items><itemname>Toner</itemname><price>920</price><quantity>1</quantity></items> <poID>ABC-CO-19282</poID><shipTo><street>	public PurchaseOrder updatePO(PurchaseOrder order)

	<items> <itemname>Toner</itemname> <price>920</price> <quantity>1</quantity> </items> <shipTo> <street>1 Main Street</street> <city>Beverly Hills</city> <state>CA</state> <zipCode>90210</zipCode> </shipTo> </tns:PurchaseOrderDocument>	1 Main Street</street><city>Beverly Hills</city><state>CA</state><zipCode>90210</zipCode></shipTo></ns3:PurchaseOrderDocument>	
Delete	DELETE /restfulwebservice-war/poservice/ABC-CO-19282 HTTP/1.0 Connection: close Content-Type: text/xml Content-Length: 0 Pragma: no-cache	HTTP/1.1 200 OK X-Powered-By: Servlet/2.5 Content-Type: text/xml Date: Fri, 21 Jul 2006 17:10:38 GMT Server: Sun Java System Application Server Platform Edition 9.1 Connection: close <?xml version="1.0" encoding="UTF-8"?>	public void cancelPO(String orderID)

Tabela 5 - Mensagens HTTP Request e HTTP Response para Purchase Order Service [11]

Num caso concreto em que o Serviço tem de indicar uma excepção, pode ser feito definindo o código de estado HTTP e a mensagem de resposta `MessageContext`. Por exemplo, uma resposta a um processo com um ID inválido pode ser implementada mediante a definição do estado HTTP 400 e incluindo o código XML do schema `POProcessingProblem.xsd`.

O exemplo de código 2 [11] ilustra esta situação.

Exemplo de Código 2 (Servidor, envio de excepção)

```
Private Source get(MessageContext mc) throws JAXBException
{
  String path = (String)mc.get(MessageContext.PATH_INFO);
  if((path.indexOf("/errortest")!= -1)||
      path.equals("") ||
      path.equals("/")){
    mc.put(MessageContext.HTTP_RESPONSE_CODE, 400);
    POProcessingProblem fault= new POProcessingProblem();
    fault.setMessage("Unable to retrieve the order associated with the
orderid you specified");
    return new JAXBSource(jc,
```

```
                        new objectFactory().createPOProcessingFault(fault));
  }
}
```

Estratégia: Implementar o verbo como parte da URI.

Para processar uma ordem de compra, faz-se uma solicitação HTTP para o URL `http://127.0.0.1:8080/RESTfulwebservice-war/poservice/acceptPO`

O serviço ou recurso interpreta o verbo na URI como a acção que deve executar. O serviço recupera os dados necessários a partir do pedido, que pode ser fisicamente um HTTP GET ou POST, e ele retorna dados baseados no *schema* `PurchaseOrder.xsd`.

Estratégia: Utilizar o método para descrever o verbo ou operação.

Para obter uma ordem de compra em que o ID é ABC123, faz-se uma solicitação HTTP usando a operação GET. Neste caso concreto, a URL solicitada seria `http://127.0.0.1:8080/RESTfulwebservice-war/poservice/ABC123`

No exemplo a seguir, uma amostra de solicitação HTTP:

```
GET /RESTful webservice-war/poservice/ABC123 HTTP/1.0
Accept-Language: en-us
Connection: close
Content-Type: text/xml
Host: 127.0.0.1:9090
```

Para cancelar uma ordem de compra em que o ID de ordem é ABC123, basta fazer uma solicitação HTTP usando a operação DELETE. Neste caso, a URL seria também `http://127.0.0.1:8080/RESTfulwebservice-war/poservice/ABC123`

A seguir, uma amostra de solicitação HTTP usando a operação DELETE:

```
DELETE /RESTful webservice-war/poservice/ABC123 HTTP/1.0
Accept-Language: en-us
Connection: close
Content-Type: text/xml
Host: 127.0.0.1:9090
```

```
Pragma: no-cache
```

Consumir Serviços RESTful

As aplicações podem aceder aos Serviços RESTful numa das duas maneiras: através de programação/ código ou através do *browser*.

Aceder aos Serviços através de Programação/ Código

JAX-WS permite que um cliente consuma Serviços Web RESTful através de programação. O principal é a interface API `javax.xml.ws.Dispatch` descrita no Exemplo de Código 3 [11].

Exemplo de Código 3 (Cliente)

```
// T é o tipo de mensagem
public interface Dispatch<T>
{
// Pedido de resposta síncrono
 T invoke(T msg);

// Pedido de resposta assíncrono
 Response<T> invokeAsync(T msg);
 Future<?> invokeAsync(T msg, AsyncHandler<T> h);

// Uma via
 void invokeOneWay(T msg);
}
```

Ao contrário do `Provider`, do lado do servidor esta API não é implementada. Em vez disso, obtêm-se uma instância do objecto `Service` como mostrado aqui:

```
service = Service.create(qname);
service.addPort(qname, HTTPBinding.HTTP_BINDING, url);
Dispatch<Source> dispatcher = service.createDispatch(new QName("", ""),
 Source.class, Service.Mode.PAYLOAD);
```

A interface `Dispatch<T>` e o método `invoke` podem aceitar e retornar quatro grandes *datatypes*:

- `Dispatch<javax.xml.transform.Source>`. Útil para o modo HTTP *binding payload;*
- `Dispatch<javax.xml.soap.SOAPMessage>`. Útil para mensagens SOAP;
- `Dispatch<javax.activation.DataSource>`. Útil para manipulação de mensagens MIME;
- `Dispatch<Object>`. Útil para *payload* com JAXB (*Binding).*

O exemplo de código 4 [11] mostra como fazer uma solicitação `POST` para `http://127.0.0.1:8080/RESTfulwebservice-war/poservice/acceptPO` com o XML como o corpo da solicitação `POST`.

Exemplo de Código 4 (Cliente, Solicitação POST)

```
private void acceptPO()
{
  Service service = Service.create(qname);
  service.addPort(qname, HTTPBinding.HTTP_BINDING, url + "acceptPO");
  Dispatch<Source> dispatcher = service.createDispatch(qname,
  Source.class, Service.Mode.MESSAGE);
  Map<String, Object> requestContext = dispatcher.getRequestContext();
  requestContext.put(MessageContext.HTTP_REQUEST_METHOD, "POST");
  Source result = dispatcher.invoke(
                            new StreamSource(newStringReader(poXML)));
  printSource(result);
}
```

O exemplo de código 5 [11] demonstra como fazer um pedido POST, mas difere do Exemplo de Código 4, em que envia e retorna objectos JAXB em vez de manipular *strings* directamente.

Exemplo de Código 5 (Cliente, Solicitação POST usando JAXB)

```
Private void acceptPOJAXB() throws JAXBException
{
  service.addPort(qname, HTTPBinding.HTTP_BINDING, url + "acceptPO");
  Dispatch<Object> dispatcher = service.createDispatch(qname, jc,
                                                  Service.Mode.PAYLOAD);
  Map<String, Object> requestContext = dispatcher.getRequestContext();
  requestContext.put(MessageContext.HTTP_REQUEST_METHOD, "POST");
  JAXBElement<PurchaseOrder> order =
              new ObjectFactory().createPurchaseOrderDocument(createPO());
  JAXBElement<PurchaseOrderStatus> response =
              (JAXBElement<PurchaseOrderStatus>)dispatcher.invoke(order);
  PurchaseOrderStatus result= response.getValue();
  System.out.println("Order id is : " + result.getOrderid());
  System.out.println("Timestamp is : " + result.getTimestamp());
}
```

O exemplo de código 6 [11] demonstra como fazer um pedido PUT usando a interface `Dispatch`.

Exemplo de Código 6 (Cliente, Solicitação PUT)

```
private void updatePO()
{
  Service service = Service.create(qname);
  service.addPort(qname, HTTPBinding.HTTP_BINDING, url);
  Dispatch<Source> dispatcher = service.createDispatch(qname,
                                                  Source.class,
                                                  Service.Mode.MESSAGE);
  Map<String, Object> requestContext = dispatcher.getRequestContext();
  requestContext.put(MessageContext.HTTP_REQUEST_METHOD, "PUT");
  Source result = dispatcher.invoke(new StreamSource(new
                                                    StringReader(poXML)));
  printSource(result);
}
```

Aceder aos Serviços Usando Browsers

Como os Serviços Web RESTful implementados no JAX-WS são expostos usando o protocolo HTTP padrão e respectivos métodos, podem ser facilmente acedidos a partir de *browsers*. Para além de usar simples pedidos `GET` e `POST` directamente a partir de *browsers*, os programadores podem alavancar as capacidades do objecto `XMLHttpRequest` da tecnologia JavaScript que a maioria dos *browsers* modernos suporta. Este é o mesmo objecto utilizado para a construção de interfaces de utilizador AJAX (UI's).

O exemplo de código 7 mostra um *script* simples que se pode usar para testar Serviços Web RESTful a partir de *browsers*. A Figura 5 mostra o visor resultante deste código.

Exemplo de Código 7 (Script Cliente para invocar todas as operações CRUD)

```
<script type=”text/javascript" language="javascript">
var http_request = false;

//Envio de solicitações CRUD ao Servidor
function makePOSTRequest(method,url, parameters)
{
  http_request = false;
  if (window.XMLHttpRequest) { // Mozilla, Safari,...
   http_request = new XMLHttpRequest();
   if (http_request.overrideMimeType) {
     // Set type accordingly to anticipated content type.
     http_request.overrideMimeType('text/xml');
   }
  } else if (window.ActiveXObject) { // IE
     try {
      http_request = new ActiveXObject("Msxml2.XMLHTTP");
     } catch (e) {
     try {
      http_request = new ActiveXObject("Microsoft.XMLHTTP");
     } catch (e) {}
    }
  }
  if (!http_request) {
   alert('Cannot create XMLHttpRequest object);
   return false;
  }
```

```
  http_request.onreadystatechange = alertContents;
  // http_request.open(method, url, true);

  if(method=='GET'){
   http_request.open(method, url+parameters, true);
   http_request.setRequestHeader("Content-type",  "text/xml");
   http_request.setRequestHeader("Content-length", parameters.length);
   http_request.setRequestHeader("Connection", "close");
   http_request.send(null);
  }
  if(method=='POST')  {
   http_request.open(method, url, true);
   http_request.setRequestHeader("Content-type", "text/xml");
   http_request.setRequestHeader("Content-length",parameters.length);
   http_request.setRequestHeader("Connection", "close");
   http_request.send(parameters);
  }
  if(method=='PUT')  {
   http_request.open(method, url, true);
   http_request.setRequestHeader("Content-type","text/xml");
   http_request.setRequestHeader("Content-length",parameters.length);
   http_request.setRequestHeader("Connection", "close");
   http_request.send(parameters);
  }
  if(method=='DELETE')  {
   http_request.open(method, url+parameters, true);
   http_request.setRequestHeader("Content-type", "text/xml");
   http_request.setRequestHeader("Content-length",parameters.length);
   http_request.setRequestHeader("Connection", "close");
   http_request.send(null);
 }
}
function alertContents()
{
 if (http_request.readyState == 4) {
   if (http_request.status == 200) {
     alert('Response received from server:\n'+http_request.responseText);
     result = http_request.responseText;
     // Turn < and > into &lt; and &gt; for displaying on the page.
     result = result.replace(/\<([^!])/g, '&lt;$1');
```

```
      result = result.replace(/([^-])\>/g, '$1&gt;');
      document.getElementById('serverresponse').innerHTML = result;
    } else {
      alert('There was a problem with the request.'
                  +http_request.responseText +' '+http_request.status);
      document.getElementById('serverresponse').innerHTML =
                                  http_request.responseText;
    }
  }
 }

 function postTheForm()
 {
  var poststr = document.myform.xmldata.value ;
  alert('Sending XML to server:\n'+poststr);
  makePOSTRequest('POST',document.myform.endpointURL.value , poststr);
 }

 function getTheForm()
 {
  var getStr = encodeURI(document.myform.xmldata.value) ;
  alert('Sending XML to server:\n'+getStr);
  makePOSTRequest('GET',document.myform.endpointURL.value , getStr);
 }

 function putTheForm()
 {
  var poststr = document.myform.xmldata.value ;
  alert('Sending XML to server:\n'+poststr);
  makePOSTRequest('PUT',document.myform.endpointURL.value , poststr);
 }

 function deleteTheForm()
 {
  var getStr = encodeURI(document.myform.xmldata.value) ;
  alert('Sending XML to server:\n'+getStr);
  makePOSTRequest('DELETE',document.myform.endpointURL.value, getStr);
 }
</script>
```

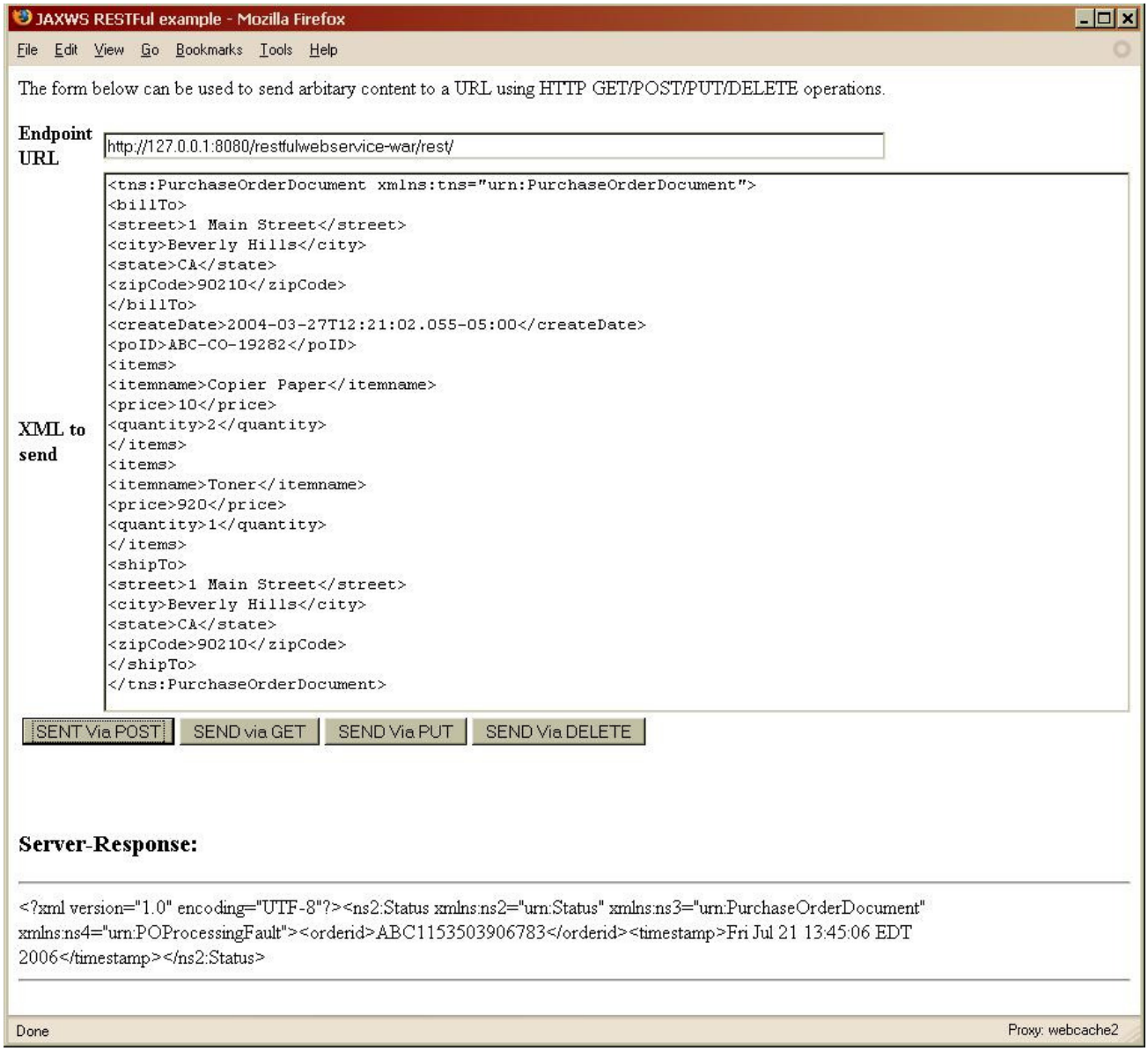

Figura 5 – Exemplo RESTfull [11]

Descrever RESTful Endpoints

Ao contrário dos Serviços Web baseados em SOAP, que têm um vocabulário para descrever o padrão de Serviço da Web através de interface WSDL, Serviços Web RESTful actualmente não têm essa gramática. Para um consumidor de Serviços para compreender o contexto e o conteúdo dos dados que devem ser enviadas e recebidas através do serviço, tanto o consumidor de serviços e o produtor de serviço devem possuir um. Este assume a forma de documentação, amostras de código e uma API que o prestador de serviços público para que os

programadores possam usar. Por exemplo, muitos serviços disponíveis na web do Google, Yahoo, Flickr, têm indicações descrevendo a forma de consumir os serviços [11].

Boas práticas indicam orientações gerais para o desenvolvimento de Serviços Web RESTful:

- Desenvolver os *schemas* XML disponíveis para os consumidores de serviços e "empacotá-los" em WAR.
- Documentar de forma clara os *inputs* e os *outputs* e condições de erro que podem surgir como resultado da invocação.

Web Application Description Language (WADL)

Um esforço de investigação da Sun Labs chamado *Web Application Description Language* (WADL) tenta resolver alguns problemas, fornecendo um meio para descrever os serviços em termos de *schemas*, métodos HTTP e o pedido ou resposta de estruturas trocadas. O *schema* no exemplo de código 8 [11] mostra uma descrição WADL para o exemplo discutido anteriormente.

Exemplo de Código 8 (Descrição WADL dos serviços)

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://research.sun.com/wadl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="urn:PurchaseOrderDocument" xmlns:ns2="urn:Status"
xmlns:ns3="urn:POProcessingFault">
  <grammars>
     <include href="POProcessingProblem.xsd"/>
     <include href="PurchaseOrder.xsd"/>
     <include href="PurchaseOrderStatus.xsd"/>
  </grammars>
  <resources base="http://127.0.0.1:8080/restfulwebservice-war/poservice">
     <resource uri="acceptPO">
        <method href="#POSTacceptPO"/>
        <method href="#GETacceptPO"/>
     </resource>
     <resource>
        <path_variable name="orderID" type="xsd:string"/>
        <method href="#retrievePO"/>
     </resource>
```

```
     <resource>
        <method href="#updatePO"/>
     </resource>
     <resource>
        <path_variable name="orderID" type="xsd:string"/>
        <method href="#cancelPO"/>
     </resource>
  </resources>

  <method id="POSTacceptPO" name="POST"
href="http://127.0.0.1:8080/restfulwebservice-war/poservice">
     <request>
        <representation element="ns1:PurchaseOrderDocument "
mediaType="text/xml"/>
     </request>
     <response>
        <representation element="ns2:Status" mediaType="text/xml"/>
     </response>
  </method>

  <method id="GETacceptPO" name="GET"
href="http://127.0.0.1:8080/restfulwebservice-war/poservice">
     <request>
        <representation element="ns1:PurchaseOrderDocument "
mediaType="text/xml"/>
     </request>
     <response>
        <representation element="ns2:Status" mediaType="text/xml"/>
     </response>
  </method>

  <method id="retrievePO " name="GET"
href="http://127.0.0.1:8080/restfulwebservice-war/poservice">
     <response>
        <representation element="ns1:PurchaseOrderDocument "
mediaType="text/xml"/>
        <fault id="POProcessingFault" status="400" mediaType="text/xml"
element="ns3:POProcessingFault"/>
     </response>
  </method>
```

```
  <method id="updatePO " name="PUT"
href="http://127.0.0.1:8080/restfulwebservice-war/poservice">
     <request>
        <representation element="ns1:PurchaseOrderDocument "
mediaType="text/xml"/>
     </request>
     <response>
        <representation element="ns1:PurchaseOrderDocument "
mediaType="text/xml"/>
     </response>
  </method>

  <method id="cancelPO" name="DELETE"
href="http://127.0.0.1:8080/restfulwebservice-war/poservice"/>
</application>
```

2.7. Qualidade dos Serviços Web

Quando se considera um Serviço Web abrangido pelos requisitos QoS (*Quality of Service*), supomos que a interface é alargada com declarações sobre QoS que podem ser associadas a toda a interface ou a operações individuais e atributos.

No caso de uma invocação de serviço, estas declarações descrevem a respectiva associação QoS com o serviço exigido pelo cliente, enquanto que a partir de uma perspectiva do fornecedor do serviço, estas declarações descrevem os requisitos QoS associados com o serviço oferecido.

QoS abrange toda uma gama de técnicas que correspondam às necessidades dos serviços entre os fornecedores de serviços e os clientes, com base nos recursos de rede disponíveis. QoS pode ser definida por propriedades não funcionais de serviços Web, tais como desempenho, confiabilidade, disponibilidade, integridade, segurança e fiabilidade. Os principais requísitos QoS que suportam os Serviços Web são descritos abaixo [10].

Disponibilidade

Disponibilidade é o aspecto da qualidade que sabe se o serviço Web está disponível ou pronto para uso imediato. Os valores mais elevados indicam que o serviço está com maior probabilidade para uso, enquanto valores menores indicam imprevisibilidade de saber se o serviço vai estar disponível num determinado momento. Também está associada a disponibilidade de tempo para a reparação (TTR). TTR representa o tempo que leva para reparar um serviço que falhou. Idealmente pequenos valores de TTR são desejáveis.

Acessibilidade

A Acessibilidade é o aspecto da qualidade de um serviço que representa o grau em que é capaz de servir um pedido Web. Pode ser expressa como uma probabilidade medida onde denota a taxa de sucesso ou possibilidade de uma bem sucedida instanciação do serviço num determinado ponto no tempo. Pode haver situações em que um Serviço Web está disponível, mas não acessível. Alta acessibilidade dos Serviços Web pode ser alcançada através da construção de sistemas altamente escaláveis. Escalabilidade refere-se à capacidade de servir constantemente os pedidos apesar das variações no volume de pedidos.

Integridade

Integridade é o aspecto da qualidade que mede a forma como o Serviço Web mantém a correcta interacção em relação à fonte. Uma correcta execução das operações do Serviço Web proporcionará uma correcta interacção. Uma operação refere-se a uma sequência de actividades tratadas como uma única unidade de trabalho. Todas as actividades têm de ser cumpridas para que a operação seja bem sucedida. Quando uma operação não é completada com sucesso, todas as alterações feitas são revertidas.

Desempenho

O desempenho é o aspecto da qualidade do Serviço Web, que é medido em termos de *throughput* e latência. Maior *throughput* e menores valores de latência representam um bom desempenho do Serviço Web. *Throughput* representa o número de solicitações ao Serviço Web num determinado período de tempo. Latência é o tempo entre o envio de um pedido e recepção da resposta.

Fiabilidade

A fiabilidade é o aspecto da qualidade de um Serviço Web que representa a capacidade de manter o serviço e a qualidade mesmo. O número de falhas por mês ou ano representa uma medida de fiabilidade de um Serviço Web. Noutro sentido, fiabilidade refere-se à segura e ordenada entrega de mensagens enviadas e recebidas pelos invocadores e fornecedores do serviço.

Conformidade

Conformidade é o aspecto da qualidade do Serviço Web em conformidade com as normas, a lei, o cumprimento das normas, e estabelece o acordo de nível de serviço. Serviços Web utilizam várias regras, tais como SOAP, UDDI e WSDL.

O cumprimento rigoroso das normas das versões correctas (por exemplo, SOAP versão 1.2), pelos fornecedores de serviços é necessário para uma correcta invocação dos Serviços Web pelos invocadores.

Segurança

A segurança é o aspecto da qualidade do Serviço Web de prestação de confidencialidade pela autenticação das partes envolvidas, criptografia de mensagens, e proporciona um controlo de acessos. A segurança tem importância acrescida porque a invocação dos Serviços Web ocorre através do serviço público da Internet. O fornecedor do serviço pode ter diferentes abordagens e níveis de prestação de serviço de segurança, dependendo do solicitador.

Interoperabilidade

Serviços Web permitem que diferentes aplicações comuniquem entre si e partilhem dados e serviços. Outras aplicações também podem utilizar os serviços. Por exemplo VB ou .NET pode falar com Serviços Web Java e vice-versa. Portanto, Serviços Web são independentes da plataforma ou tecnologia.

Protocolo Padronizado

Serviços Web utilizam padrões da indústria, protocolos padronizados para a comunicação. Todas as quatro camadas (*Service Transport, XML Messaging, Service Description and*

Service Discovery) usam o protocolo SOAP. Esta padronização de protocolo dá ao negócio muitas vantagens, como por exemplo, uma ampla gama de opções, a redução do custo devido à concorrência e ao aumento da qualidade.

Expor a função para a rede

Um Serviço Web é uma unidade de código, que pode ser gerido remotamente, invocado através de HTTP. Por isso, Serviços Web permitem que esteja disponível a funcionalidade do seu código existente na rede. Uma vez que está exposta na rede, outra aplicação pode usar a funcionalidade da aplicação.

Baixo custo de comunicação

Serviços Web usam SOAP sobre o protocolo HTTP para a comunicação, de modo que é possível usar uma ligação normal à Internet. Esta solução é muito menos dispendiosa, em comparação com soluções proprietárias. Além de SOAP sobre HTTP, os Serviços Web também podem ser aplicados noutros mecanismos de transporte fiável como FTP.

2.8. Serviços Web e os Processos de Negócio

Actualmente, grande parte das organizações existentes a nível mundial, estruturam a sua actividade com base no conceito de processos de negócio. Uma vez que os sistemas de informação apresentam uma importância indiscutível como suporte aos processos da organização, surge a necessidade de sistemas cada vez mais flexíveis e de fácil integração. Neste contexto, tornam-se importantes os Serviços Web como tecnologia de suporte aos processos de negócio, cujas características preenchem cada vez mais os requisitos referidos. No entanto, a representação dos processos deve ser feita de forma independente da tecnologia que os implementa, factor que presentemente nem sempre é satisfeito.

A Internet surge actualmente como uma plataforma essencial de integração de sistemas e aplicações, visto que consiste num conjunto de normas e padrões aceites universalmente, o que permite a comunicação entre dois computadores de uma forma simples e automatizada. No entanto, a Internet não fornece todos os mecanismos necessários para que esta integração seja possível, pois apenas fornece o conjunto de protocolos necessários à comunicação.

Os Serviços Web, como tecnologia que permite a interoperabilidade universal utilizando normas e padrões comuns, permitem consequentemente a integração de diversas aplicações nos domínios do B2C (*Business to Consumer*), B2B (*Business to Business*), entre outros.

Inicialmente, os Serviços Web começaram por se basear no conjunto de padrões, já descritos, o SOAP que especifica o formato de mensagens, o HTTP como protocolo de transporte de dados e o UDDI como um mecanismo de publicação e pesquisa de serviços.

A importância da modelação de processos de negócio e sua representação num determinado contexto organizacional, tem sido cada vez mais reconhecida como forma de controlar e optimizar o *workflow*/ *dataflow* de uma determinada organização.

Os Serviços Web surgem actualmente como uma das tecnologias de maior importância e das mais inovadoras, que permitem integrar diversas aplicações, o que permite às organizações disponibilizarem vários serviços a entidades externas.

Os Serviços Web são actualmente uma tecnologia que suporta inúmeros processos de negócio, quer estes representem interacções intra ou inter-empresariais, dado que a sua importância tem crescido exponencialmente nos últimos anos.

Assim sendo, têm surgido várias propostas no que se refere à modelação de processos de negócio utilizando Serviços Web, tais como o BPEL4WS (*Business Process Execution Language for Web Wervices*), BPML (*Business Process Modeling Language*) e DAML (*DARPA Agent Markup Language*).

No entanto, não é vantajoso nem correcto do ponto de vista do modelo de negócio, existirem detalhes ou conceitos tecnológicos na definição de um processo de negócio utilizando uma destas linguagens, tal como acontece actualmente.

Tendo em conta este factor, foi proposto em um modelo de representação de processos de negócio cuja base tecnológica assente em Serviços Web e que permite definir detalhes técnicos aquando da representação de um determinado processo, e que se descreve seguidamente.

2.9. Camadas de Processos de Negócio

Tendo como objectivo o desenvolvimento de um modelo de representação comum a todos os Serviços Web, ou seja, que permita representar um processo de negócio independentemente da tecnologia que os implemente, surge a dificuldade de determinar quais os conceitos comuns a um Serviço Web [38].

O modelo descrito abaixo é uma forma de representação de um processo de negócio cujo suporte a nível tecnológico é feito recorrendo aos Serviços Web. Este modelo é composto por três camadas distintas (ver Fig. 8):

- Processos de Negócio: camada onde apenas se identifica os vários intervenientes do processo de negócio e respectivas trocas de informação entre eles (*workflow/ dataflow*);
- Serviços Web: camada com o objectivo de representar os serviços disponibilizados por cada interveniente do Processo de Negócio descrito na camada anterior, descrição de detalhes técnicos ou implementações específicas;
- Tecnológica: camada responsável por descrever os detalhes técnicos e implementações adequadas para a descrição, pesquisa e execução de um serviço;

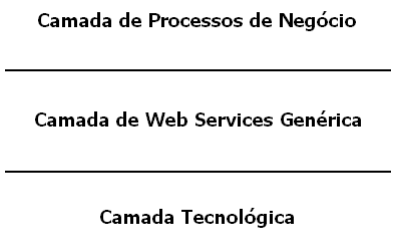

Figura 6 – Representação das três camadas dos Processos de Negócio [38]

Processos de Negócio

Esta primeira camada é constituída por três elementos cujas relações entre si permitem definir o *workflow/ dataflow* associado a um determinado processo de negócio (Fig. 9).

Os elementos definidos no contexto desta secção e envolvidos nesta camada são:

- *Actor*: elemento que suporta a execução de uma actividade de um Processo de Negócio; pode ser um conjunto de elementos humanos e/ ou tecnológicos. Neste caso, os actores limitam-se aos sistemas que são importantes no âmbito dos Serviços Web devido à existência das noções de *requester* e *provider* de um serviço;

- *Activity*: elemento que representa uma actividade de um processo de negócio. A actividade deve ser entendida como um conjunto de acções/ operações que consomem/ produzem um ou mais recursos;

- *Resource*: elemento que representa um recurso que é consumido ou produzido por uma actividade, através de uma relação de CRUD (*Create, Read, Update e Delete*)

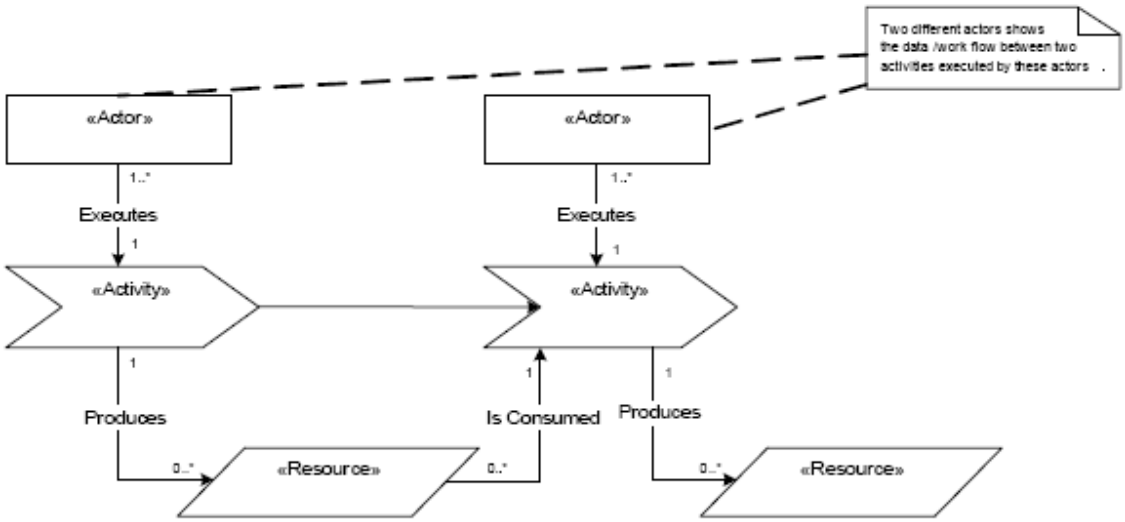

Figura 7 - Camada de Processos de Negócio - Representação do fluxo de informação entre actividades executadas por diferentes actores [38]

Serviços Web

Esta camada representa as entidades que disponibilizam um ou mais serviços, bem como a descrição dos mesmos de uma forma genérica, a informação trocada e as operações executadas (Fig. 10).

A visão estática pretende ser uma representação da entidade que fornece um ou mais serviços. A utilidade desta visão justifica-se com o facto de existirem situações em que não é relevante representar quem invoca o serviço, mas apenas que o disponibiliza.

A visão dinâmica, tal como o nome indica, permite representar a invocação de um determinado serviço. Nesta visão são representadas ambas as entidades que participam nesta interacção (quem invoca e quem disponibiliza determinado serviço).

Os elementos definidos para esta camada são os seguintes:

- *ServiceBlock*: representa uma estrutura aplicacional que disponibiliza um conjunto de serviços ou que invoca uma operação de um serviço fornecido por outro *ServiceBlock* (esta distinção é feita recorrendo à classe *ServiceRole*).
- *ServiceRole*: representa uma classe que tem por objectivo especificar o papel desempenhado por um *ServiceBlock*. Esta classe é caracterizada como *Requester* ou *Provider* caso o *ServiceBlock* disponibilize serviços ou invoque uma operação.
- *Service*: representa um serviço que é disponibilizado por um determinado *ServiceBlock*. É, também, o elemento que agrega operações relacionadas.
- *Operation*: representa a operação que é invocada por um *Service Requester*. Tem uma determinada funcionalidade associada e consome ou produz determinada informação representada pelo elemento *Document*.
- *Document*: representa a informação trocada resultante da invocação de uma operação no contexto de um serviço.

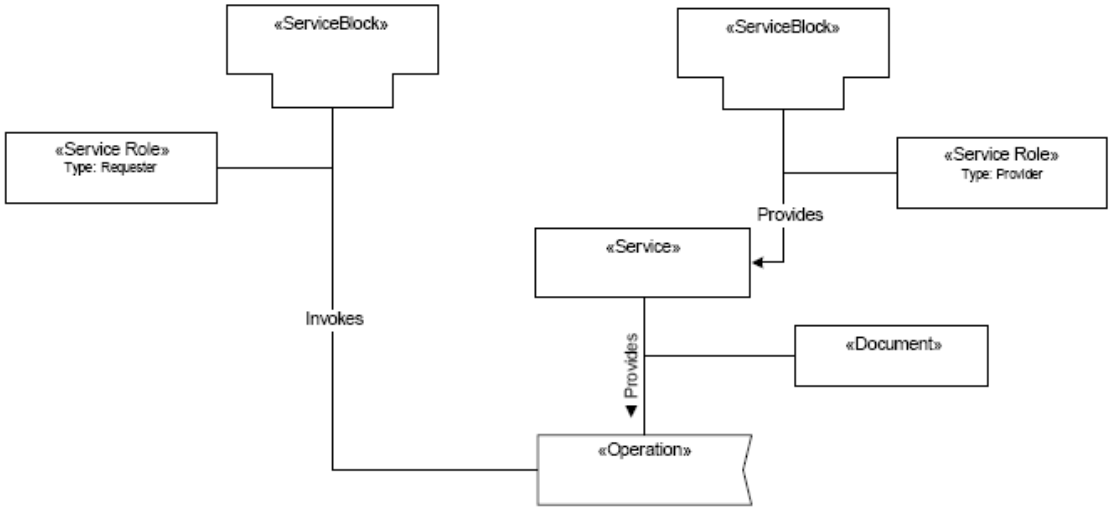

Figura 8 – Camada de Serviços Web [38]

Tecnológica

Esta camada foi desenvolvida com o objectivo de caracterizar os detalhes técnicos associados à execução de um determinado serviço (ver Fig. 11), detalhes esses de que as camadas superiores se pretendem abstrair.

Consiste em diversos elementos que, entre outras características, permitem representar o local físico onde se executa o serviço, a operação executada, as mensagens trocadas, o protocolo de transporte a ser utilizado, etc.

Os elementos definidos para esta camada são os seguintes:

- *Operation*: representa a operação que é executada por um fornecedor de serviços e invocada por um solicitante.
- *PortType*: representa a interface abstracta de um serviço e define um conjunto de operações com características comuns. Deste ponto de vista, diz-se que um *PortType* é a classe agregadora de operações.
- *Binding*: descreve como a interface abstracta do serviço *PortType* é mapeada em protocolos específicos de formato e transmissão de dados.
- *Port*: indica a localização específica de um serviço, associando uma ligação a um endereço de rede (URI).
- *Message*: representa a informação trocada entre as duas entidades intervenientes na execução de um Serviço Web. Pode ser caracterizada em mensagem de *Input, Output ou Fault* (esta permite indicar uma falta na execução de uma determinada operação).
- *Parameter*: representa um parâmetro de uma mensagem. Cada mensagem pode ter um ou mais parâmetros sejam do tipo primitivo (string, int, etc) ou uma estrutura de dados definida em XML.

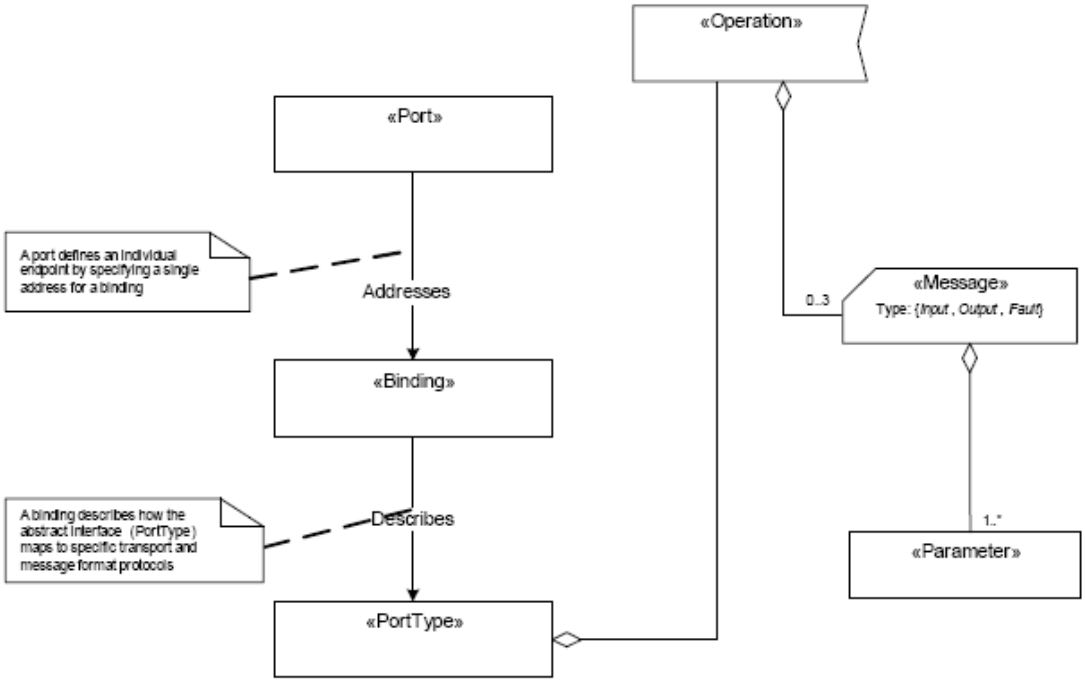

Figura 9 – Camada Tecnológica [38]

Mapeamento entre Camadas

Tendo por base os pontos anteriores obtém-se o modelo final com o respectivo mapeamento de camadas que é apresentado na Fig. 12

Do mapeamento da camada de Processos de Negócio para a camada de Serviços Web destaca-se o facto de um *ServiceBlock* constituir um elemento dedicado à disponibilização de um serviço (e execução de respectivas operações do mesmo) numa visão microscópica de um actor (*Actor*) e, tendo em conta os conceitos de recurso (*Resource*) e documento (*Document*) pode ser ainda realçada a relação entre estes mesmos, ainda que pertençam a camadas distintas.

Relativamente ao mapeamento da camada Serviços Web para a camada Tecnológica surge a rastreabilidade entre o conceito de documento (*Document*), que representa a informação trocada aquando da invocação do serviço e o conceito de mensagem (*Input Message e Output Message*), com o mesmo objectivo de representação mas transposto para a camada Tecnológica. Uma vez que um serviço (*Service*) está associado um *Port* (ou *Endpoint*), surge nova rastreabilidade entre estes dois conceitos pertencentes às camadas distintas.

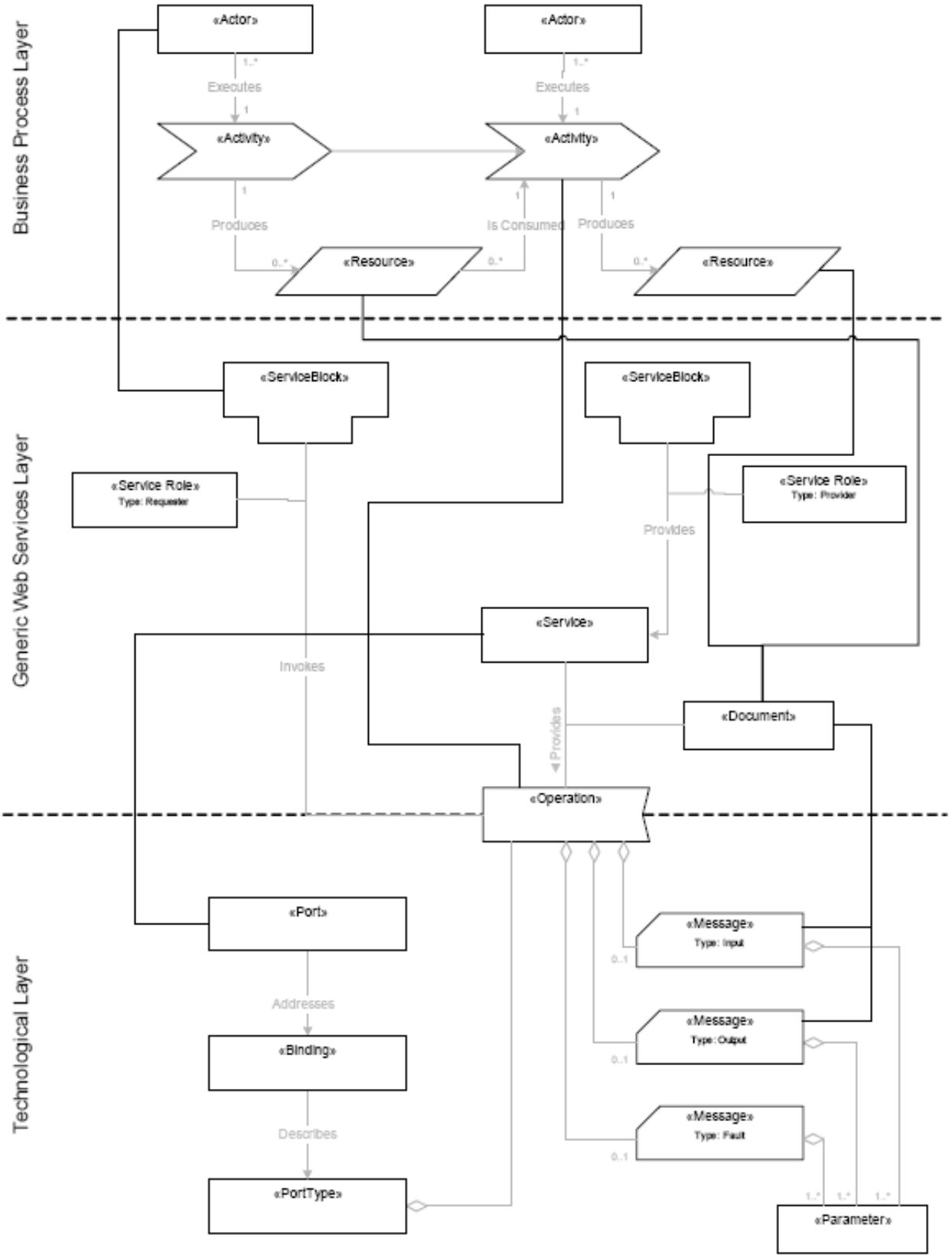

Figura 10 - Mapeamento entre Camadas [38]

O modelo acima descrito pretende evitar que sejam confundidos conceitos relacionados com processos de negócio e conceitos relacionados com a tecnologia que é utilizada para os implementar.

Relativamente à camada de Serviços Web, esta foi desenvolvida através dos conceitos que se revelaram fundamentais para descrever um determinado serviço e com o objectivo de criar rastreabilidade entre os conceitos desta camada e da camada de processos de negócio.

Quanto à camada Tecnológica, esta procura representar aspectos técnicos relacionados com a descrição e invocação dos serviços utilizados pelo processo de negócio. Para isso foi considerada como base de descrição de qualquer Serviço Web, a linguagem WSDL que actualmente é usada na grande maioria dos Serviços Web existentes.

Modelo de Representação de um Processo de Negócio

A necessidade um modelo que permitisse a representação de um processo de negócio cujo suporte tecnológico fossem Serviços Web, surge na sequência de estudos anteriores, que demonstram que as linguagens de modelação de processos de negócio confundem conceitos inerentes ao processo de negócio com os conceitos inerentes à tecnologia que os implementa.

Conclui-se actualmente que a base do suporte tecnológico dos Serviços Web faz sempre uso de especificações tais como WSDL para descrever a interface dos serviços, SOAP para descrever o formato das mensagens trocadas e HTTP como protocolo de transporte utilizado.

Devido à sua simplicidade de implementação, descrição, execução e integração com outras aplicações, os Serviços Web são actualmente a mais popular implementação de uma arquitectura SOA (*Service Oriented Arquitecture*), embora existam outras que fornecem benefícios semelhantes.

Há ainda que referir que na sua origem, as arquitecturas baseadas em Serviços Web continham algumas lacunas que dificultavam a sua integração em alguns casos, tal como a necessidade em garantir a confidencialidade das mensagens trocadas através de mecanismos de encriptação ou controlar a execução de um Serviço Web composto por diversos intervenientes recorrendo a mecanismos transaccionais.

Tendo em conta que a tecnologia dos Serviços Web é relativamente recente e ainda está em crescimento, o modelo acima não abrange todos os aspectos relacionados com esta tecnologia.

2.10. Potencialidades e Capacidades

Hoje, a Web tem terabytes de informação disponível, mas escondida nos computadores. É um paradoxo que a informação é presa dentro de páginas HTML, formatado em esotéricas maneiras. A chamada Web 3.0, que é susceptível de ser um precursor da real Web Semântica vai mudar isso. O que queremos dizer com "Web 3.0" é que os grandes sites da Web vão ser transformados em Serviços Web - e irão expor eficazmente a sua informação para o mundo.

A transformação vai acontecer numa de duas maneiras. Alguns sites vão seguir o exemplo da Amazon, del.icio.us e Flickr e vão oferecer as suas informações através de uma API REST. Outros tentarão manter as suas informações proprietárias, mas vão ser abertos através de mashups[1] criados usando serviços como o Dapper, Teqlo e Yahoo! Pipes. O resultado será que as informações não estruturadas irão dar lugar a informações estruturadas - abrindo o caminho para uma computação mais inteligente.

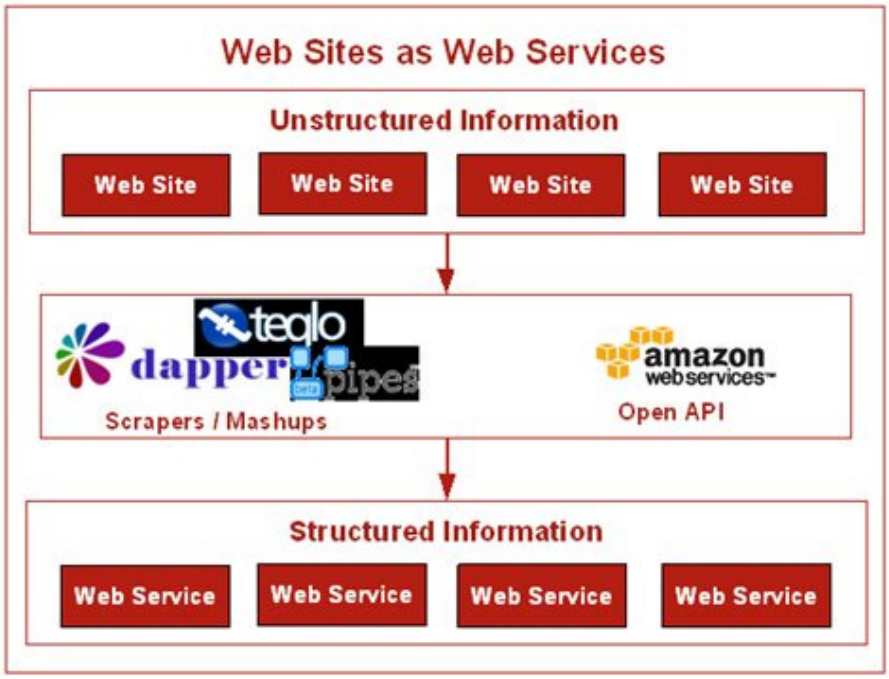

Figura 11 – Estruturação da informação da Amazon, Dapper, Teclo e Yahoo Pipes [3]

A Amazon pretende reinventar-se por expor a sua própria infra-estrutura através de um conjunto de distintas API's. Um dos primeiros Serviços Web abertos pela Amazon foi o

[1] Um *mashup* é um website ou uma aplicação web que usa conteúdo de mais de uma fonte para criar um novo serviço completo. O conteúdo usado em *mashups* é tipicamente código de terceiros através de uma interface pública ou de uma API.

serviço de E-Commerce. Este serviço abriu acesso para a maioria dos itens no catálogo de produtos da Amazon. A API é bastante rica, permitindo a manipulação de utilizadores, listas e carrinhos de compras. No entanto, a sua essência é a capacidade de pesquisa dos produtos da Amazon.

Del.icio.us, também é famoso como uma das primeiras empresas a abrir um subconjunto de funcionalidade API no seu Web site. Muitos serviços seguiram-se, dando origem a uma verdadeira cultura da API. Esta página mostra quase 400 API's organizadas por categoria. No entanto, apenas uma fracção dessas API's disponibilizam informação. A API del.icio.us oferece um serviço diferente da Amazon, porque não abre a base de dados del.icio.us para o mundo. Aquilo que se pretende fazer é permitir mashups autorizados a manipular as informações do utilizador armazenadas no del.icio.us. Por exemplo, uma aplicação pode adicionar um *post*, ou actualizar uma *tag* [3].

Outros métodos de codificação de conteúdo para mashups incluem Web feeds (exemplo: RSS ou Atom), Javascript e widgets que podem ser entendidas como mini aplicações web, disponíveis para serem incorporadas a outros sites.

3. Especificações Normativas

Existe uma variedade de especificações inerentes aos Serviços Web. Estas especificações estão em diferentes graus de maturidade e são apoiadas por diversas entidades. São vulgarmente denominadas de "*WS*-*". As especificações trabalham conjuntamente para fornecer protocolos interoperáveis para a Segurança, a Confiabilidade das mensagens e Transações em sistemas interligados. As especificações são suportadas por XML e SOAP.

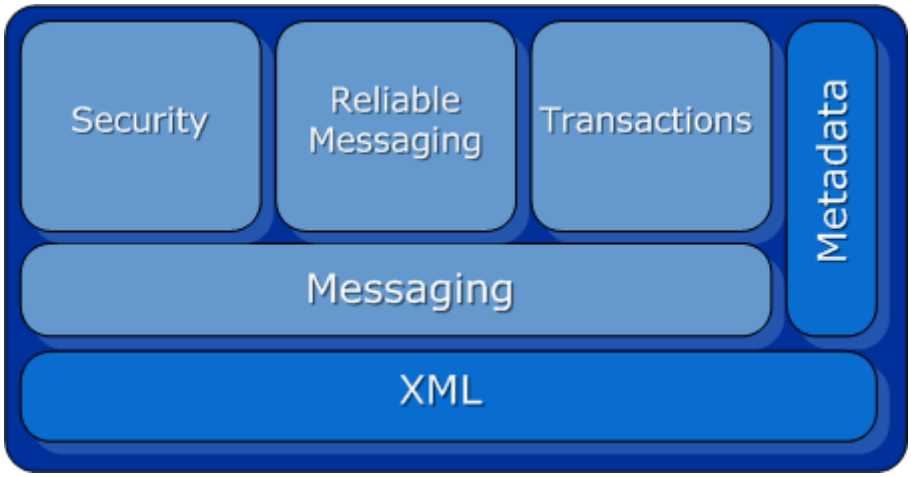

Figura 12 – Arquitectura das especificações dos Serviços Web [29]

WS-Specifications

As especificações dos Serviços Web trabalham em conjunto para fornecer/ permitir protocolos interoperáveis para a segurança, troca de mensagens eficazes e eficientes, e transacções em sistemas diversos. As especificações situam-se acima das normas XML e SOAP.

As *WS-Specifications* constroem uma arquitectura combinada de modo a formar um ambiente complexo de aplicações de Serviços Web. Diferentes fornecedores, como BEA, IBM, Microsoft, RSA Security e SAP, uniram forças para lançar os alicerces para aplicações de Serviços Web seguras e fiáveis, que suportam diferentes tecnologias e múltiplos participantes.

A maioria das especificações estão ainda na fase de projecto e estão sujeitas a alterações, mas já existem produtos que suportam determinadas características. *WS-Specifications* foram concebidas para serem usadas com o SOAP 1.1 e 1.2 [29].

3.1. Especificações ao nível das Mensagens

No parágrafo que se segue, são apresentadas as especificações ao nível das Mensagens:

- WS-Routing
- WS-Addressing
- MTOM
- WS-Enumeration
- WS-Eventing
- WS-Transfer
- SOAP-over-UDP
- SOAP 1.1 Ligação for MTOM 1.0

WS-Routing

O caminho de uma mensagem SOAP desde o primeiro remetente ao destinatário final faz-se através de um conjunto de intermediários opcionais e pode ser expressa no cabeçalho SOAP usando a *Web Services Routing Protocol*. O caminho inverso também pode ser expresso desta forma. A especificação permite a troca de várias mensagens padrões *request/response*, *message acknowledgement* e *peer-to-peer conversation.*

WS-Addressing

WS-Addressing disponibiliza mecanismos de transporte neutros para endereçar os serviços e mensagens. Especificamente, *WS-Addressing* define os elementos XML para identificar os *endpoints* do serviço Web e guarda a identificação do *endpoint*. Esta especificação permite que o apoio na transmissão das mensagens de uma forma neutra através de redes que incluem nós de processamento, tais como de *endpoints*, *firewalls* e *gateways*. Além da construção de referências de *endpoints*, a *WS-Addressing* fornece informações de cabeçalhos da mensagem (*headers*).

WS-Addressing foi submetido para o W3C em Agosto de 2004.

MTOM (Message Transmission Optimization Mechanism)

MTOM descreve um mecanismo de optimização para a transmissão de uma mensagem SOAP através da recodificação de partes da mensagem e ainda apresentando um *Infoset* XML para a aplicação SOAP.

MTOM também descreve um mecanismo de inclusão, que opera de forma obrigatória e independente, além de uma ligação específica para o HTTP.

WS-Enumeration

WS-Enumeration permite que uma aplicação possa pedir itens de uma lista de dados que está suportada num Serviço Web. Desta forma, o *WS-Enumeration* é útil para ler os *logs* de eventos, filas de mensagens ou outras colecções de dados.

WS-Enumeration foi submetido para o W3C em Março de 2006.

WS-Eventing

Uma infra-estrutura de troca de mensagens assíncrona é um alicerce para aplicações distribuídas complexas. *WS-Eventing* permite a utilização do *Observer Design Pattern*[2] para Serviços Web. *WS-Eventing* descreve como construir um evento orientado de troca de mensagens padrão usando conceitos *WS-Addressing*, permitindo que os serviços Web actuem como fontes de eventos para os assinantes. Define as operações necessárias para gerir as subscrições de fontes de eventos, bem como a forma como as mensagens de eventos reais são construídas.

WS-Eventing foi submetido para o W3C em Março de 2006.

[2] É um padrão de observação no qual um objecto, chamado de sujeito, mantém uma lista dos seus dependentes, chamados observadores, e notificá-los automaticamente sobre quaisquer mudanças de estado, geralmente, chamando um dos seus métodos. É usado principalmente para implementar sistemas de manutenção de eventos distribuídos.

WS-Transfer

WS-Transfer define como invocar um simples conjunto de métodos (GET, POST, PUT e DELETE) usando SOAP. Um protocolo de aplicação pode ser construído para realizar estas operações sobre os recursos.

WS-Transfer foi submetido para o W3C em Março de 2006.

SOAP-over-UDP

A especificação *SOAP-over-UDP* define uma ligação entre o SOAP e datagramas de utilizadores, incluindo, padrões de mensagens, requisitos de endereçamento e considerações de segurança.

SOAP 1.1 Binding for MTOM

Esta especificação detalha as modificações necessárias para a transmissão de mensagens SOAP [MTOM] e *XML-binary Optimized Packaging* [XOP] necessárias para utilizar essas tecnologias com sucesso com a SOAP 1.1. As especificações MTOM e XOP foram originalmente especificadas pelo W3C com ligações normativas para SOAP 1.2. No entanto, apesar da intenção de as usar num contexto SOAP 1.1, nenhuma ligação formal foi definida.

3.2. Especificações ao nível da Segurança

O SOAP não garante a transmissão segura de mensagens, o que coloca riscos nos remetentes e nos destinatários das mensagens. Embora as tecnologias tradicionais de segurança como o SSL e HTTPS podem resolver parcialmente este problema, encriptando as mensagens entre os dois pontos, estas tecnologias não podem garantir segurança ponto-a-ponto ao longo do percurso de um Serviço Web num complexo sistema distribuído multi-camadas. Além disso, estas tecnologias de segurança ponto-a-ponto são baseadas no protocolo de transporte específico, como a camada TCP/ IP para SSL e HTTP para HTTPS. SOAP é um protocolo de transporte de mensagens independente dos Serviços Web e a capacidade e aplicação dos Serviços Web seria limitada se a sua segurança dependesse destas tecnologias de transporte. Como resultado, OASIS desenvolveu a especificação WSS, com o objectivo de fornecer protecção ao nível das mensagens entre os dois pontos (clientes e Serviços Web) através da integridade, da confidencialidade e da autenticidade das mensagens. WSS usa a arquitectura

extensível SOAP incorporando informações relacionadas à segurança (tokens de segurança e assinaturas) no *header* SOAP, sem afectar os dados armazenados no *body*.

Esta arquitectura permite ainda que WSS integre com SOAP como um plug-in e ainda manter a modularidade e extensibilidade SOAP para outros fins. Cada vez mais, os Serviços Web estão a suportar o padrão WSS. Enquanto o WSS aumenta a segurança dos Serviços Web, a preocupação deve ser orientada e focada no desempenho, pois questões de *overhead* podem ocorrer:

- Maiores tempos de CPU para processar elementos/ operações relativos ao WSS, tanto nos clientes como nos serviços;
- Maior latência de rede, uma vez que as mensagens são maiores devido ao conteúdo adicional WSS.

Assim, o pretendido é uma avaliação de desempenho do WSS numa abordagem experimental. Um simples Serviço Web foi testado para permitir a troca de uma variedade de mensagens de diferentes tamanhos entre o Serviço Web e seus clientes.

3.2.1. Arquitectura

A Figura 13 define a arquitectura de segurança de Serviços Web. Esta arquitectura mostra os componentes chave de segurança, suas características e as relações entre elas. O principal objectivo está no WSS, o elemento fundamental na arquitectura, e que foi especificado pela OASIS como *WS-Security* [16].

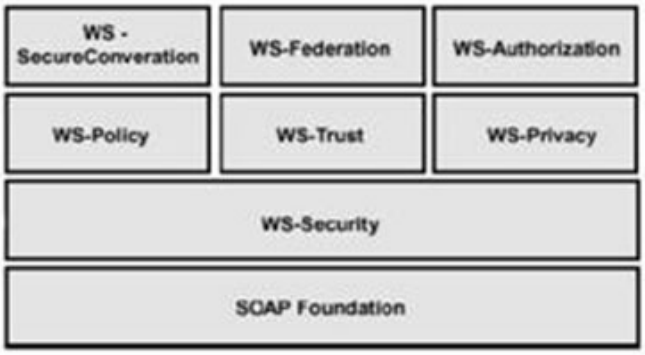

Figura 13 - Arquitectura de Segurança de Serviço Web [16]

SOAP é o núcleo do protocolo de mensagens de Serviços Web. Uma mensagem SOAP é construída com um envelope, que consiste num *header* e num *body*. Enquanto o *body* é obrigatório e normalmente é utilizado para transportar dados ao nível da aplicação, o *header* fornece um mecanismo flexível para compor esquemas para extensões. O WSS aproveita esta flexibilidade para fornecer mecanismos de segurança que melhoram a integridade da mensagem e a confidencialidade da mesma. Por exemplo, permite que os *tokens* de segurança, que carregam as credenciais de segurança para autenticação, sejam anexados à mensagem e especifiquem de que forma os binários dos *tokens* são codificados.

No parágrafo que se segue, são apresentadas as especificações ao nível da Segurança:

- WS-Security
- WS-Security: SOAP Message Security
- WS-Security: UsernameToken Profile
- WS-Security: X.509 Certificate Token Profile
- WS-SecureConversation
- WS-SecurityPolicy
- WS-Trust
- WS-Federation
- WS-Federation Active Requestor Profile
- WS-Federation Passive Requestor Profile
- WS-Security: Kerberos Ligação
- Web Single Sign-On Interoperability Profile
- Web Single Sign-On Metadata Exchange Protocol

WS-Security (WSS)

A segurança é um requisito para a adopção de Serviços Web em muitas aplicações, especialmente nas críticas, em que a integridade e a confidencialidade das mensagens devem ser garantidas. O *Web Service Security Language*, ou *WS-Security* é uma base para a implementação de uma ampla gama de soluções de segurança. Em Abril de 2004, a *Organization for the Advancement of Structured Information Standards* (OASIS) ratificou o WS-Security como uma norma sob o nome *OASIS Web Service Security (WSS)*. A assinatura

e encriptação de mensagens SOAP, bem como a propagação de *tokens* de segurança são suportadas pela *WS-Security*. *WS-Security* utiliza a linguagem XML, *Signature* e *XML Encryption*, normas W3C.

Quase todas as Especificações WS podem ser usada em conjunto com *WS-Security*. Por exemplo, um elemento `WSP: Policy` dentro de um cabeçalho SOAP deverá ser assinado para impedir a manipulação ilícita. *WS-SecureConversations* e *WS-Trust* são camadas acima de *WS-Security*.

O SAML (*Security Assertion Markup Language*) é uma norma emergente para a troca de dados relativamente a autenticações e autorizações. O SAML possibilita que os utilizadores transportem as suas informações entre diferentes Serviços Web. Isto é importante para as aplicações que pretendem integrar um número de Serviços Web para criar uma aplicação unida.

WS-Security: SOAP Message Security

A *WS-Security* descreve aperfeiçoamentos para mensagens SOAP para proporcionar qualidade de protecção através da integridade e da confidencialidade das mensagens, bem como, da autenticação única de mensagem.

WS-Security oferece, também, um mecanismo extensível para associar os *tokens* de segurança de mensagens.

Além disso, o *WS-Security* descreve como codificar binários de *tokens* de segurança, mais concretamente, os certificados X.509 e etiquetas Kerberos, bem como, a forma de incluir chaves criptografadas.

WS-Security: UsernameToken Profile

Este documento descreve o uso de *UsernameToken*[3] com a especificação *WS-Security*. O elemento `<UsernameToken>` transporta um nome de utilizador e, opcionalmente as informações de senha. Este tipo de *token* é utilizado para transportar informações de autenticação básica.

[3] Descreve como um consumidor do Serviço Web pode fornecer um UsernameToken como um meio de identificação do solicitante através de um "*username*" e, opcionalmente, usando uma senha para autenticar a identidade do fornecedor do serviço web.

WS-Security: X.509 Certificate Token Profile

Esta especificação descreve o uso da plataforma de autenticação X.509 com a especificação *WS-Security*. *Web Services Security (WSS): X.509 Certificate Token Profile* descreve como usar certificados X.509 em conjunto com a especificação de segurança *WSS: SOAP Message Security.* Mais especificamente, aborda como um Serviço Web pode usar certificados X.509 como meio de fornecer autenticação entre um fornecedor de serviços e o solicitador.

WS-SecurityPolicy

WS-SecurityPolicy indica a política definida na especificação *WS-Policy* que aplica à *WS-Security, WS-Trust* e *WS-SecureConversation*, oferecendo mecanismos para representar as capacidades e necessidades de Serviços Web como políticas.

As políticas podem ser usadas para exigir mais segurança genérica, através de atributos como camada de transporte segura `<TransportBinding>`, segurança ao nível da mensagem ou *timestamps* `<AsymmetricBinding>` e atributos específicos, como *tokens*.

WS-Trust

WS-Trust é uma extensão da especificação *WS-Security*. A especificação permite a emissão, renovação e validação de *tokens* de segurança, bem como com as formas de estabelecer, avaliar a presença e a confiança entre os participantes numa troca de mensagens seguras.

WS-SecureConversation

Num contexto em que mais de duas mensagens são trocadas, utilizar a especificação *WS-SecureConversation* pode ser mais apropriado do que usar *WS-Security*. *WS-SecureConversation* define as extensões que se baseiam em *WS-Security* para fornecer comunicações seguras. Especificamente, define mecanismos para a criação e partilha de contextos de segurança e derivação de chaves de sessão a partir de contextos de segurança.

WS-Federation

Cenários complexos de Serviços Web podem expandir-se ao longo de vários Serviços Web, localizados em diferentes domínios de segurança. É incómodo para um utilizador ou uma aplicação Web aferir e validar a sua identidade em cada serviço. Uma federação permite

virtualmente o acesso a Serviços Web em diferentes domínios. *WS-Federation*, não só proporciona a propagação transparente de identidades, mas prevê, também, a propagação arbitrária de atributos.

WS-Federation Active Requestor Profile

WS-Federation Active Requestor Profile define a identidade, a autenticação e mecanismos de autorização da federação definidos em *WS-Federation*, que são usados por invocadores activos, tais como aplicações SOAP activas.

WS-Federation Passive Requestor Profile

WS-Federation Passive Requestor Profile descreve a identidade, a autenticação e mecanismos de autorização da federação definidos em *WS-Federation*, que são usados por invocadores passivos, como navegadores da Web para fornecer serviços de identidade. Os solicitadores passivos deste perfil são limitados ao protocolo HTTP.

WS-Security: Kerberos Binding

WS-Security: Kerberos Binding descreve como usar as especificações de segurança dos Serviços Web com Kerberos[4].

Web Single Sign-On Interoperability Profile

Web Single Sign-On Interoperability Profile define um perfil de interoperabilidade que permite a utilização quer da *Liberty Identity Federation*[5] quer de fornecedores baseados em entidades *WS-Federation* para interagir com um serviço.

[4] Kerberos é o nome de um protocolo de rede, que permite comunicações individuais seguras e identificadas, numa rede insegura.

[5] Em Julho de 2002, a Liberty Alliance lançou a sua primeira especificação pública, a Liberty Identity Federation (ID-FF) 1.0. A Liberty Identity Federation permite que os consumidores e utilizadores de serviços baseados na Internet e aplicações de comércio eletrónico se autentiquem, numa rede ou domínio, a partir de qualquer dispositivo e, em seguida, visitar ou consumir serviços de vários sites.

Web Single Sign-On Metadata Exchange Protocol

Web Single Sign-On Metadata Exchange Protocol define como um serviço pode consultar um fornecedor por forma a obter os metadados que descrevem o protocolo de processamento de identidade suportado por esse fornecedor.

3.3. Especificações ao nível das Mensagens Fidedignas/ Confiáveis

No parágrafo que se segue, são apresentadas as especificações ao nível das Mensagens Fidedignas/ Confiáveis:

- WS-ReliableMessaging
- WS-Reliable
- WS-RM Policy

WS-ReliableMessaging

Uma sequência de mensagens pode ser entregue confiadamente usando *WS-ReliableMessaging*. O receptor reconhece as mensagens recebidas e pede para a redistribuição de mensagens perdidas.

Esta especificação descreve um protocolo que permite que as mensagens sejam entregues de forma confiável entre aplicações distribuídas, na presença de componentes de software, sistema ou falhas de rede. O protocolo é descrito nesta especificação como uma forma de transporte independente que lhe permite ser implementado usando tecnologias de transporte de redes diferentes. Para suportar a interoperabilidade dos Serviços Web, é definida uma ligação SOAP dentro desta especificação.

WS-Reliability

Esta especificação é um revés para *WS-ReliableMessaging*. *WS-Reliability* foi criada pela Sun Microsystems, Oracle, Sonic Software e outros. Os seus objectivos - entrega garantida, eliminar a duplicação de mensagens e a ordenação de mensagem - são idênticos às do *WS-ReliableMessaging*. *WS-Reliability* foi apresentado à norma OASIS como um padrão aberto.

Ambas as normas devem convergir para uma só, porque é necessária uma especificação para mensagens confiáveis.

WS-RM Policy

Esta especificação descreve uma definição que potencia a plataforma *WS-Policy* para permitir uma troca de determinada mensagem de forma confiável.

WS-ReliableMessaging e *WS-RM* Policy juntaram-se ao Comité Técnico OASIS WS-RX[6] em Junho de 2005.

3.4. Especificações ao nível das Transacções

No parágrafo que se segue, são apresentadas as especificações ao nível das Transacções:

- W-Coordination
- WS-AtomicTransaction
- WS-BusinessActivity
- WS-Transaction

WS-Coordination

WS-Coordination é uma plataforma extensível para o estabelecimento de uma coordenação entre Serviços Web e coordenadores. Podem ser definidos diferentes tipos de coordenação. Cada tipo de coordenação pode ter protocolos de coordenação múltipla. Podem ser criados contextos utilizados para transacções ou segurança e associados com mensagens. Um contexto contém uma referência a um registo de serviço. *WS-Transaction* apoia a especificação *WS-Coordination*.

Esta especificação descreve uma estrutura para fornecer protocolos que coordenam as acções de aplicações distribuídas. Permite o processamento de transacções, fluxo de trabalho e outros sistemas de coordenação para esconder os seus protocolos proprietários e para operar num

[6] Protocolo de troca de mensagens confiáveis entre dois Serviços Web, através do contínuo desenvolvimento dos Serviços Web através da especificação *Web Services Reliable Messaging*, bem como definir um mecanismo pelo qual os Serviços Web expressem o apoio de mensagens confiáveis, assim como outros parâmetros úteis nesta especificação.

ambiente heterogéneo. Além disso, esta especificação descreve uma definição da estrutura do contexto e os requisitos para a propagação de contexto entre os serviços que colaboraram.

WS-AtomicTransaction

A especificação *WS-AtomicTransaction* descreve as operações atómicas. Porque os recursos envolvidos na transacção são bloqueadas durante a operação, a operação deve abranger um período curto. Se mais de dois recursos participam numa operação, o protocolo *two phase commit protocol (2PC)* é usado para coordenar os participantes.

Os protocolos de monitores de propriedade de transacção podem ser acondicionados de forma a operar em ambientes de Serviços Web.

Esta especificação fornecerá a definição do tipo de transacção de coordenação atómica para ser usada com o quadro de coordenação extensível descrito na especificação *WS-Coordination*. A especificação define três protocolos de acordo com uma coordenação específica para o tipo de transacção de coordenação atómica: *completion*, *volatile two-phase commit*, e *durable two-phase commit*.

WS-BusinessActivity

Transacções de longa execução são diferentes das transacções de curta execução, tais como transacções de dados. Em ambientes de Serviços Web, uma transacção pode abranger vários dias, portanto bloquear os recursos não é recomendado. As acções são executadas imediatamente, e as alterações referentes aos dados são permanentes. Na eventualidade de um erro, as acções são tomadas para compensar as modificações já autorizadas.

A especificação *WS-BusinessActivity* que suporta transacções de longa execução irá substituir a parte II da especificação obsoleta *WS-Transaction*. Neste momento, a especificação *WS-BuisnessActivity* ainda não existe.

Esta especificação fornece a definição do tipo de coordenação de actividades de negócios que deve ser usada com o quadro de coordenação extensível descrito na especificação *WS-Coordination*. A especificação definirá dois protocolos específicos para o tipo de coordenação de actividades de negócio: *BusinessAgreementWithParticipantCompletion* e *BusinessAgreementWithCoordinatorCompletion*.

WS-Coordination, *WS-AtomicTransaction*, e *WS-BusinessActivity* juntaram-se ao Comité Técnico OASIS WS-TX[7], em Novembro de 2005.

WS-Transaction

A especificação *WS-Transaction* é uma versão obsoleta das especificações *WS-AtomicTransaction* e *WS-BuisnessActivity*.

3.5. Especificações ao nível dos Metadados

No parágrafo que se segue, são apresentadas as especificações ao nível dos Metadados:

- WSDL
- WSDL 1.1 Bnding Extension for SOAP 1.2
- WS-Policy
- WS-PolicyAssertions
- WS-PolicyAttachment
- WS-Discovery
- WS-Inspection
- WS-MetadataExchange
- WS-MTOMPolicy

WSDL

Sigla para *Web Service Description Language*, uma linguagem de formato XML usado para descrever as capacidades de um Serviço Web como colecções de comunicação capazes de trocar mensagens.

WSDL define uma gramática baseada em XML para descrever serviços de rede como um conjunto de terminais que aceitam informações baseadas em documentos ou procedimentos. As operações e mensagens são descritas de maneira abstracta e estando vinculadas a um

[7] WS-TX (Web Services Transaction) é um conjunto de especificações XML projectado para permitir a utilização de protocolos abertos, para transacções seguras e confiáveis em toda a Web.

protocolo de rede concreto e formato de mensagem para definir um ponto final. Os parâmetros concretos relacionados são combinados em terminais abstractos (serviços).

WSDL é suficientemente extensível para permitir a descrição dos parâmetros e das mensagens, independentemente dos formatos das mensagens ou protocolos de rede usados. No entanto, este documento apenas descreve como usar o WSDL em conjunto com SOAP 1.1, HTTP GET/ POST, e MIME.

WSDL 1.1 Binding Extension for SOAP 1.2

WSDL 1.1 Binding Extension for SOAP 1.2 descreve como indicar num documento WSDL 1.1 que o serviço usa SOAP 1.2. Deve ser considerada como uma solução transitória para descrever SOAP 1.2 de serviços baseados na Web.

WS-Policy

A *WS-Policy* define capacidades, requisitos de sistema e políticas. Por exemplo, uma política pode indicar que um Serviço Web só aceita pedidos que contenham uma assinatura válida ou que o tamanho de uma mensagem não deve ser excedido. Não se encontra definida nesta especificação como uma política pode ser obtida. *WS-MetadataExchange* e *WS-PolicyAttachment* especificam como as políticas são acessíveis através de mensagens SOAP ou associadas com documentos XML e WSDL.

WS-Policy foi submetido para o W3C em Abril de 2006.

WS-PolicyAttachment

A descrição das definições ou capacidades de um Serviço Web é expressa em *WS-Policy*. Deve estar associada com o Serviço Web, a fim de considerar a política antes de invocar o serviço. *WS-PolicyAttachment* especifica como as políticas podem ser associados com o XML e WSDL ou registados em UDDI. Também define a política de aplicação específica associada à totalidade ou parte de um *Port* WSDL quando expostos a partir de uma aplicação específica.

WS-PolicyAttachment foi submetido para o W3C em Abril de 2006.

WS-MetadataExchange

Metadados em documentos WSDL, esquemas XML e *WS-Policy* são úteis para invocar um Serviço Web com êxito. Como obter informações sobre os parâmetros deste serviço é o tema da especificação *WS-MetadataExchange*.

WS-MetadataExchange define como os metadados associados a um terminal podem ser representados como recursos *WS-Transfer*, como os metadados podem ser incorporados em *WS-Addressing Endpoint References*, e como os metadados podem ser recuperados a partir de um terminal.

WS-Discovery

Esta especificação define um protocolo *multicast* de descoberta para localizar serviços. Por norma, os pedidos são enviados para um grupo multicast, e os serviços de destino que correspondem, retornam uma resposta directamente ao invocador. Para dimensionar um grande número de terminais, o protocolo define o comportamento de supressão selectiva se um proxy está disponível na rede. Para minimizar a necessidade de pesquisa, os serviços que desejam ser descobertos enviam um aviso, quando entram e saem da rede.

WS-Inspection

Serviços Web que são disponibilizados por um site podem ser publicado usando um documento *WS-Inspection*. O documento *WS-Inspection* fornece descrições feitas para o serviço como descrições de serviço WSDL ou registos UDDI. *WS-Inspection* não compete com UDDI, pelo contrário, é uma forma de descrição que trabalha em conjunto com UDDI.

WS-MTOMPolicy

Esta especificação descreve políticas ou regras de domínio específicas que indicam o terminal de suporte optimizado MIME multipart/ relacionadas com mensagens SOAP.

3.6. Especificações ao nível do XML

No parágrafo que se segue, são apresentadas as especificações ao nível do XML:

- XML

- Namespaces in XML
- XML Information Set

XML

Sigla de *eXtensible Markup Language*. Permite criar as suas próprias etiquetas, permitindo a definição, transmissão, validação e interpretação de dados entre aplicações e entre organizações.

Namespaces in XML

Namespaces XML fornecem um método simples para a qualificação de elementos e atributos usados em *Extensible Markup Language* (XML), associando-as com espaços identificados por referências de URI.

XML Information Set

XML Information Set (XML Infoset) descreve um modelo de dados abstrato de um documento XML em termos de conjuntos de itens de informação. A sua finalidade é fornecer um conjunto coerente de definições XML para o uso de outras especificações.

3.7. Especificações ao nível da Gestão

No parágrafo que se segue, são apresentadas as especificações ao nível da Gestão:

- WS-Management
- WS-Management Catalog
- WS-ResourceTransfer
- WS-ResourcePlataforma
- WS-Notification

WS-Management

WS-Management foi publicada pela primeira vez como uma especificação pública em 10 de Outubro de 2004. Esta é a terceira publicação conjunta da especificação.

Esta especificação descreve um protocolo geral baseado em SOAP para sistemas, tais como, PC's, servidores, dispositivos, Serviços Web e/ ou aplicações.

WS-Management contribuiu para DMTF em Agosto de 2005 e ratificado como padrão preliminar em Agosto de 2006.

WS-Management Catalog

Esta especificação define os formatos de metadados padrão para *WS-Management*. O catálogo é usado para descobrir a gestão dos recursos disponíveis, tais como adaptadores de rede ou unidades de disco.

WS-ResourceCatalog

Esta especificação define um catálogo para organizar e classificar os recursos de gestão. O catálogo destina-se a anunciar os recursos acessíveis via *WS-Management*, WSDM e outras especificações de gestão, incluindo os seus protocolos subjacentes. Esta especificação pode ser composta com outros dados descritos na especificação do Serviço Web.

WS-ResourceTransfer

Esta especificação estende as operações da *WS-Transfer*, adicionando a capacidade de operar em fragmentos de representação da gestão de recursos.

WS-Resource Plataform (WSRF)

Uma abordagem diferente para a arquitectura dos Serviços Web *"representational state transfer"* geridos do lado do servidor como recursos. O *WS-Resource Plataform* introduz um estado, sob a forma de recursos, de Serviços Web baseados em SOAP. WSRF introduz um modelo padrão que descreve como aceder aos recursos com Serviços Web.

WS-Notification

WS-Notification explora o *WS-Resource Plataform* e Serviços Web.

O objectivo da *WS-Notification* é definir um conjunto de especificações que padronizam a forma como os Serviços Web interagem com "Notificações" ou "Eventos", concretamente especifica como usar mensagens *Publish-Subscribe* com os Serviços Web. Esta especificação

permite de forma padronizada divulgar informações para um conjunto de outros Serviços Web, sem ter conhecimento prévio deles.

3.8. Especificações Adicionais

No parágrafo que se segue, são apresentadas as especificações adicionais:

- *Remote Shell Web Services Protocol*
- *Business Process Specifications*

Remote Shell Web Services Protocol

Esta especificação descreve um conjunto de extensões da especificação *WS-Management* para aceder ao processamento da linha de comandos.

Business Process Specifications

Business Process Execution Language for Web Services (BPEL4WS) fornece um meio para especificar formalmente os processos de negócios e protocolos de interacção. Ao fazê-lo, estende o modelo de interacção dos Serviços Web e permite suportar a transacções de negócios. BPEL4WS define um modelo de integração interoperável que deve facilitar a expansão do processo de integração automatizado.

3.9. Especificações ao nível de Perfis

No parágrafo que se segue, são apresentadas as especificações ao nível de Perfis:

- *Devices Profile*
- *WS-I Basic Profile*

Devices Profile

Para permitir um nível base de interoperabilidade entre dispositivos e Serviços Web, o *Devices Profile* procura as especificações internamente e define como usá-las por forma a permitir a troca de mensagens seguras, a descoberta, a descrição e os eventos em implementações com limitações de recursos.

WS-I Basic Profile

Este documento define o *WS-I Basic Profile 1.0* e consiste numa série de especificações não proprietárias, juntamente com esclarecimentos e emendas para as especificações que promovem a interoperabilidade.

Sobre este ponto, existe ainda um grande fosso entre os recursos abundantes do CORBA e Serviços Web baseados em SOAP. *WS-Specifications* irá ajudar a preencher essa lacuna. No entanto, a maioria das WS-*Specifications* estão ainda em fases iniciais. A especificação *WS-Security* é essencial no alicerce para a maioria das aplicações e está bastante completa. Uma especificação muito promissora é *WS-Addressing*, porque é necessária para passar um parâmetro de referência de um serviço para outro.

As maiores empresas na área dos Serviços Web apoiam as especificações para que exista uma elevada probabilidade de uma grande quota de mercado.

3.10. Aplicação prática

No parágrafo abaixo, é apresentada uma descrição das características e métodos dos Serviços Web:

Service description: nos Serviços Web, a descrição de serviços contém a seguinte informação:

```
<?xml version="1.0" encoding="UTF-8"?>
 <definitions name="HelloService"
targetNamespace="http://www.foo.com/wsdl/PrinterService.wsdl" xmlns=...>
 <message name="PrintRequest">
 <part name="data" type="xsd:string"/>
 </message>
 <message name="PrintResponse">
 <part name="jobID" type="xsd:string"/>
 </message>
 <portType name="Print_PortType">
 <operation name="print">
 <input message="tns:PrintRequest"/>
 <output message="tns:PrintResponse"/>
 </operation>
 </portType>
```

```
 <binding name="Print_Binding"
type="tns:Print_PortType">
 ...
 </binding>
 <service name="Print_Service">
 <documentation>
 WSDL pour service impression
 </documentation>
 <port binding="tns:Print_Binding" name="Print_Port">
 <soap:address location="http://www.printhost:8080/printsvca"/>
 </port>
 </service>
 </definitions>
 <?xml version="1.0" encoding="iso-8859-1"?>
```

- **definitions:** nome do serviço e espaço de nome
- **types:** definição de tipos de dados complexos
- **message:** descrição de uma mensagem (resposta). Esta descrição contêm o nome da mensagem e o número de partes que descrevem os parâmetros ou retornam valores.
- **portType:** descrição do método que combina várias mensagens, por exemplo, um pedido e uma resposta.
- **binding:** descrição do protocolo de comunicação da mensagem.
- **service:** localização e nome do serviço (URL).

Service publication: Os principais métodos que são disponibilizados pelo UDDI para publicar informações são:

- **save_service:** publica um tipo de serviço.
- **save_business:** publica um fornecedor.

Os principais métodos que permitem remover informações do *registry* são:

- **delete_service:** remove um tipo de serviço.
- **delete_business:** remove um fornecedor.

Service discovery: UDDI disponibiliza diferentes métodos de procura:

- **find_service:** retorna informação acerca dos serviços disponibilizados por um negócio.
- **find_business:** retorna informação acerca de um ou mais fornecedores.

Os métodos de procura suportam *queries* de expressões regulares. Estes métodos retornam chaves que podem ser usadas posteriormente para obter informação detalhada e os valores podem ser retornados por ordem, segundo um dos seguintes critérios: alfabética, data de registo, e disponibilidade certificada.

Service object creation policies: A criação de todas as regras/ políticas de utilização devem ser suportadas por um fornecedor.

Service notifications: UDDI pode notificar clientes acerca das alterações no registo, nomeadamente o tipo de serviços e negócios. As notificações incluem adicionar, remover, e modificar.

Service release: Dado que os Serviços Web suportam diferentes protocolos de comunicação, as políticas/ regras podem ser implementadas pelos fornecedores.

4. Desempenho de Serviços Web

SOAP permite comunicação cliente-servidor entre diferentes entidades em diferentes plataformas e linguagens, característica que torna a utilização deste protocolo, uma solução flexível. SOAP elimina, também, o problema com *firewalls* e *gateways*, pois trabalha sobre os protocolos usados na internet como HTTP e SMTP. No entanto, a flexibilidade desta técnica traz problemas quando o factor desempenho é considerado. Por usar XML, o SOAP atribui um processamento adicional na construção e na interpretação das mensagens.

***testes de desempenho**. 1. testes elaborados com o intuito de avaliar um sistema ou componentes através de requisitos de desempenho específicos.*

ISO/IEC 24765, Systems and Software Engineering Vocabulary.

4.1. Características

No parágrafo abaixo, são descritas as características dos Serviços Web e suas implicações no desenvolvimento e implementação dos mesmos:

Reprodutibilidade

A reprodutibilidade é a base de avaliação de desempenho. Esta característica envolve uma série de medidas e permite que os resultados entre os diferentes utilizadores que executam o mesmo programa de teste sejam comparáveis. Os resultados da comparação resultam não só da técnica utilizada em processos de testes, por forma a não serem afectados pelo sistema de *inputs*.

Escalabilidade

No processo de avaliação, a escalabilidade de dados pode mudar para se adequar ao ambiente de teste ou para obter um resultado optimizado. Estes requisitos exigem ferramentas de teste para serem compatíveis com a carga do servidor com diferentes sistemas de suporte.

Universalidade

O custo para produzir uma ferramenta de testes de desempenho é muito caro. Neste caso, uma plataforma de testes pode ser reutilizada em diferentes ambientes para diferentes objectivos. Para tal, o modelo de dados, das características da operação e as medidas de desempenho devem ser personalizadas de forma eficiente.

Portabilidade

A solução deve ser económica. Tal implica a monitorização do sistema e ferramentas de análise para reduzir a perturbação do mesmo, e assim minimizar a degradação do desempenho. Além disso, para comparar o desempenho dos Serviços Web em diferentes servidores, é necessário que o programa de testes possa ser portável, e, portanto, facilmente movido de um sistema para outro.

Não-intromissão

A solução deve ser tolerante a alterações nas configurações do sistema. Estas alterações não devem causar deterioração para a precisão do desempenho do sistema e não deve induzir em erros. Por exemplo, os parâmetros intrusivos (como serviços web externos, tabelas de dados adicionais, etc) são inseridos para satisfazer os requisitos modificados, sem introduzir qualquer influência nos resultados de desempenho.

4.2. Plataforma

O objectivo mais importante dos testes de desempenho de Serviços Web é medir a capacidade de resposta dos serviços. Assim, a aplicação web, a capacidade dos servidores, a capacidade da base de dados, o custo de armazenamento e as comunicações de rede serão objectos de análise [37].

È importante que a plataforma de testes permita aos utilizadores uma estrutura de testes de desempenho configurável, na medida que permite definir dinamicamente o modelo de dados, personalizar a escalabilidade da base de dados, configurar as características de transacção, as interacções de Serviços Web e medidas de desempenho para atingir objectivos concretos.

No exemplo analisado, o módulo de clientes, usa um método de operação de rede em que um cliente simula um grande número de clientes simultâneos na chamada dos serviços disponíveis.

O módulo de servidor de aplicação contém os Serviços Web em teste e os Serviços Web de apoio externo, em que, cada um é configurado como um *plug-in*.

No servidor de aplicação, há vários tipos de Serviços Web predefinidos, como novo cliente do Serviço Web, mudança de pagamento, criação de encomenda, etc.

No módulo de base de dados, o modelo de dados, incluíndo tabelas e dependências de atributos pode ser personalizado, e a inicialização da escalabilidade dos dados podem ser redimensionadas de acordo com a topologia da dependência.

Nas tabelas, incluindo a tabela de clientes, a tabela de endereços e outras, as relações entre as tabelas são, também, distribuídas de acordo com o valor de referência.

Uma plataforma de testes deve:

- Ser independente do aplicativo;
- Ser fácil de ampliar, manter e perpetuar;
- Permitir tanto os testes dirigidos por palavra-chave, bem como scripts de dados.

O modelo de dados é uma aplicação específica programada no perfil de configuração do teste, que deve ser modificado para acomodar características variáveis dos dados e fins de teste. Um modelo de dados bem definido é a base dos testes de desempenho.

4.3. Medições de desempenho

As medições de desempenho podem ser classificadas em dois tipos básicos, uma baseada no tempo e outra com base em instruções executadas por unidade de tempo. Habitualmente,

utilizam-se, duas medidas, tempo de resposta (TR) e interacções de serviços web por segundo (SIPS).

Cada interacção do Serviço Web bem sucedida tem um valor de tempo de resposta. É definido como: TR = T2 - T1.

T1 é o tempo medido antes do primeiro *byte* da interacção do Serviço Web da mensagem SOAP enviada pelos clientes ao SUT.

T2 é o tempo medido após o último *byte* da interacção do Serviço Web da mensagem SOAP recebido pelos clientes do SUT. Ambos são medidos no cliente.

4.4. Testes de Desempenho de Serviços Web na Práctica (JAVA)

Antes de iniciar testes de desempenho fiáveis e robustos a um Serviço Web deve ser configurado um ambiente de testes, tendo em conta as orientações abaixo.

4.4.1. Pré-requisitos

De início, algumas tarefas iniciais devem ser executadas. Estas tarefas dependem dos protocolos de transporte e segurança que são implementados pelo Serviço Web em teste.

Protocolos a usar:

- HTTP: Este método de transporte é suportado por defeito, nenhuma configuração adicional é necessária, para permitir o transporte via HTTP
- SSL: O espaço de trabalho deve conter os ficheiros Java™ “key store” (JKS) que são necessários para a autenticação.
- Java Message Service (JMS): A sintaxe Web Services Description Language (WSDL) deve ser compatível com os requisitos do produto.

4.4.2. Criação de Testes

Quando o teste é gerado, são chamadas mensagens XML (XML schema definition (XSD)). Durante este processo, os campos obrigatórios são criados e as opções padrão são assumidas, podendo ser alteradas na parametrização inicial.

4.4.3. Mecanismos de Autenticação

Os testes de Serviços Web suportam mecanismos de autenticação simples ou duplos:

- **Autenticação Simples:** Neste caso, o teste precisa de saber se um certificado que é emitido pelo servidor pode ser usado no transporte HTTP. Não é necessária a configuração de uma key store. Se a opção *Always trust* já estiver seleccionada, não precisa configurar uma *trust key store*.

- Se se pretende verificar se o certificado que é emitido pelo servidor corresponde ao certificado correcto, pode configurar-se uma relação de confiança (*trust key store*). Configurar uma relação de confiança desactiva a opção *Always trust* permitindo ao teste descobrir os certificados que são enviados pelo servidor.

- **Autenticação Dupla (client-side):** Neste caso, o teste deve fornecer um certificado para o servidor de acordo com a autoridade dada pelo servidor. Para isso, pode associar-se a chave que contém o certificado que deve ser produzido na secção chave.

Por exemplo, para definir a configuração de um Serviço Web usando HTTPS e um mecanismo de autenticação duplo, devem executar-se as seguintes tarefas:

- Criar uma configuração *Secure Sockets Layer* (SSL) ao nível do nó de raiz dos testes, que inclui a opção *Always trust* (todos os certificados do servidor serão aceites) e uma chave com a sua *password*. A chave irá conter os certificados para voltar ao servidor dentro do contexto da autenticação do cliente.

- Criar outra configuração do protocolo HTTP que usa esta configuração SSL.

- Definir o nome desta nova configuração do protocolo na exibição de protocolos de cada chamada de mensagem que precisa de usá-lo.

Os Serviços Web podem ser testados com certificados digitais, tanto em protocolos SSL e SOAP. Os certificados digitais devem estar contidos em *Java Keystore* (JKS) [8]. Ao lidar com ficheiros de armazenamento de chaves, deve ser definida a senha necessária para aceder às chaves, tanto no editor de segurança como no editor de teste. Por questões de segurança SOAP poderá ter de ser fornecido um nome explícito para a chave e fornecer uma senha de acesso às chaves privadas no *key store*.

4.4.4. SSL

O *Java Runtime Environment* (JRE) tem de suportar o nível de encriptação exigido pelo certificado digital que for seleccionado. Por exemplo, não pode ser usado um certificado digital que requer encriptação a 256-bit com um JRE que só suporta encriptação a 128-bit. Por defeito, a plataforma está configurada com encriptação restrita ou de força limitada. Para utilizar algoritmos de encriptação menos restritos, deve ser utilizada a política de ficheiros ilimitada

4.4.5. Anexos dos Serviços Web

Os testes de Serviços Web suportam mensagens SOAP com Anexos (SWA) e mecanismos anexos de optimização para transmissão de mensagens (*Message Transmission Optimization Mechanism* (MTOM)).

Para usar anexos dos Serviços Web, devem ser adicionadas as seguintes bibliotecas Java para o JRE:

- `Mail.jar`
- `Activation.jar`

Estes ficheiros são fornecidos com o *plugin* de protocolo de testes de desempenho dos Serviços Web.

[8] Keystore é uma class Java que representa uma biblioteca de chaves e certificados. Keystore é armazenada num ficheiro binário de formato .jks (Java Key Store), contendo as chaves públicas e privadas, protegidas por uma frase chave.

4.4.6. Limitações

Os anexos não são compatíveis com o transporte de *Java Message Service* (JMS). O envelope é enviado directamente usando codificação UTF-8.

Nem todos os algoritmos de segurança podem ser sempre entregues com todas as implementações de máquinas virtuais. Neste caso, será necessário adicionar as bibliotecas necessárias.

O editor de teste deve mostrar o envelope como reflectida no documento XML. No entanto, os algoritmos de segurança consideram o envelope como um binário. Portanto, deve ser definida a configuração de segurança SOAP para que as mensagens de entrada e saída estejam correctamente encriptadas mas continuem a ser decifradas no interior do teste.

Outra limitação é que *Arrays* não são suportadas.

4.4.7. Desempenho

O desempenho dos utilizadores virtuais depende da implementação da aplicação. Para um transporte HTTP, foram testados 900 utilizadores em simultâneo no Windows e 600 em Linux. Para o JMS, o máximo é de 100 utilizadores simultâneos, embora este número possa variar devido à aplicação assíncrona do JMS. Para além destes valores, podem ocorrer erros de ligação e a taxa de transacção vai diminuir.

4.4.8. Verificar a sintaxe WSDL dos Serviços Web JMS

Vários fornecedores *Java Message Service* (JMS) variam na sintaxe utilizada para descrever os Serviços Web. Antes de testar os Serviços Web JMS, deve garantir-se que os ficheiros WSDL cumprem os requisitos da ferramenta.

Para verificar a sintaxe dos ficheiros WSDL, deverão ser executados os seguintes passos:

1. No explorador de projecto ou explorador de teste da ferramenta IDE, localizar e abrir o ficheiro WSDL para o serviço JMS que pretende testar. Se necessário, pode importar um ficheiro WSDL a partir do sistema de ficheiros, clicando em File> Import> File System.

2. Garantir que os critérios abaixo estejam na sintaxe do ficheiro WSDL.
    - Namespace:
      `xmlns:jms="http://schemas.xmlsoap.org/wsdl/jms/"`
    - Ligações SOAP estão definidas como:
      `transport="http://schemas.xmlsoap.org/soap/jms"`
    - Transporte JMS é definido como um URL ou como um elemento `jms:address`
3. Se o ficheiro WSDL não é igual, editar o ficheiro, em seguida, guardá-lo e fechá-lo.

Por exemplo, um JMS definido como URL assemelha-se ao seguinte:

```
<soap:address
location="jms:/queue?jndiConnectionFactoryName=UIL2ConnectionF
actory;
          jndiDestinationName=queue/testQueue;
          initialContextFactory=org.jnp.interfaces.NamingConte
          xtFactory;
          jndiFornecedorURL=9.143.104.47"/>
```

Um JMS definido como um endereço, assemelha-se ao seguinte:

```
<jms:address destinationStyle="queue"
             jndiConnectionFactoryName="myQCF"
             jndiDestinationName="myQ"
             initialContextFactory="com.ibm.NamingFactory"
             jndiFornecedorURL="iiop://something:900/">
</jms:address>
```

4.4.9. Teste Manual a um Serviço Web

Para criar um teste a Serviços Web sem gravar, basta adicionar elementos de teste necessários e editar manualmente os detalhes do teste no editor.

Antes de começar, dever-se-à verificar se o Controlador do Agente está em execução no computador local.

Os testes são armazenados em projectos de teste de desempenho, que são projectos de Java. Deve ser criado um projecto de teste de desempenho antes de se criar um teste.

Verificar se existe um ficheiro WSDL válido. Garantir que os ficheiros WSDL usam a sintaxe adequada para o ambiente de teste.

Se estiverem a ser usados protocolos de segurança *Secure Sockets Layer* (SSL), deve ser garantida a existência de chaves válidas.

Se a segurança SOAP estivem em uso, deve ser assegurado um ambiente configurado com as bibliotecas e ficheiros de configuração apropriada.

4.5. Teste de Desempenho de Serviços Web na Práctica (ASP.NET)

Quando são efectuadas chamadas para serviços Web XML a partir de uma aplicação ASP.NET, poderão detectar-se problemas de contenção, desempenho fraco e bloqueios. Os clientes podem reportar que os pedidos deixaram de responder (ou "bloquearam") ou demoraram muito tempo a serem executados. Se se suspeitar de um bloqueio, poderá reciclar-se o processo de trabalho. Poderão receber-se as seguintes mensagens no registo de eventos de aplicações [14].

Se se estiver a utilizar o Microsoft IIS 5.0 (*Internet Information Services*), recebem-se as seguintes mensagens no registo de eventos de aplicações:

- Tipo de evento: Erro
- Origem do evento: ASP.NET 1.0.3705.0
- Categoria do evento: Nenhuma
- ID do evento: 1003
- Data: 5/4/2003
- Hora: 6:18:23 PM
- Utilizador: N/D
- Computador: <Nomedocomputador>
- Descrição: aspnet_wp.exe (PID: <xxx>) was recycled because it was suspected to be in a deadlocked state.
  It did not send any responses for pending requests in the last 180 seconds.

Quando se utiliza o IIS 6.0, recebem-se as seguintes mensagens no registo de eventos de aplicações:

- Tipo de evento: Aviso
- Origem do evento: W3SVC-WP
- Categoria do evento: Nenhuma
- ID do evento: 2262
- Data: 5/4/2003
- Hora: 1:02:33 PM
- Utilizador: N/D
- Computador: <Nomedocomputador>
- Descrição: O ISAPI 'C:\Windows\Microsoft.net\Plataforma\v.1.1.4322\aspnet_isapi.dll' comunicou estar danificado devido à seguinte razão: 'Bloqueio total detectado'.

Se estivermos a utilizar o IIS 6.0, recebemos as seguintes mensagens no registo de eventos do sistema:

- Tipo de evento: Aviso
- Origem do evento: W3SVC
- Categoria do evento: Nenhuma
- ID do evento: 1013
- Data: 5/4/2003
- Hora: 1:03:47 PM
- Utilizador: N/D
- Computador: <Nomedocomputador>
- Descrição: Um processo que fornecia serviços ao agrupamento de aplicações 'DefaultAppPool' excedeu os limites de tempo durante o encerramento.
- ID do processo '<xxxx>'.

Também podemos receber a seguinte mensagem de erro de excepção quando efectuamos uma chamada para o método `HttpWebRequest.GetResponse`:

"System.InvalidOperationException: There were not enough free threads in the ThreadPool object to complete the operation."

Também se pode receber a seguinte mensagem de erro de excepção no browser:

`"HttpException (0x80004005): Request timed out."`

Nota: Estas notas aplicam-se às aplicações que efectuam pedidos **HttpWebRequest** directamente.

Este problema poderá ocorrer porque o ASP.NET limita o número de *threads* de trabalho e de threads de porta de conclusão que poderão ser utilizados por uma chamada para executar pedidos.

Normalmente, uma chamada para um Serviço Web utiliza uma *thread* de trabalho para executar o código que envia o pedido e uma *thread* de porta de conclusão para receber a chamada de retorno do Serviço Web. No entanto, se o pedido for redireccionado ou necessitar de autenticação, a chamada poderá utilizar até duas *threads* de trabalho e duas *threads* de porta de conclusão. Por conseguinte, poderá esgotar o *ThreadPool* gerido quando ocorrerem múltiplas chamadas de Serviços Web ao mesmo tempo.

Por supor que o *ThreadPool* está limitado a 10 *threads* de trabalho e todos os 10 *threads* de trabalho estão actualmente a executar código que aguarda a execução de uma chamada de retorno. A chamada de retorno poderá nem ser executada porque os itens de trabalho em fila de espera para o *ThreadPool* são bloqueados até que uma *thread* seja disponibilizada.

Outra origem possível de contenção é o parâmetro `maxconnection` utilizado pelo espaço de nomes System.Net para limitar o número de ligações. Geralmente, este limite funciona como previsto. No entanto, se muitas aplicações tentarem efectuar muitos pedidos para um único endereço IP ao mesmo tempo, as *threads* poderão ter de aguardar por uma ligação disponível.

Para resolver estes problemas, podem ajustar-se os seguintes parâmetros no ficheiro Machine.config:

- *maxWorkerThreads*
- *minWorkerThreads*
- *maxIoThreads*
- *minFreeThreads*

- *minLocalRequestFreeThreads*
- *maxconnection*
- *executionTimeout*

Deve, então, proceder-se da seguinte forma:

- Limitar o número de pedidos ASP.NET que podem ser executados ao mesmo tempo para aproximadamente 12 por CPU.
- Permitir que as chamadas de retorno de Serviços Web utilizem *threads* livremente no *ThreadPool*.
- Seleccionar um valor adequado para o parâmetro `maxconnections`. Basear a sua selecção no número de endereços IP e *AppDomains* utilizados.

Nota: A recomendação para limitar o número de pedidos ASP.NET em 12 por CPU é um pouco arbitrária. No entanto, na maioria das aplicações este limite funciona correctamente.

`maxWorkerThreads` e `maxIoThreads`

O ASP.NET utiliza as seguintes duas definições de configuração para limitar o número máximo de *threads* de trabalho e de *threads* de conclusão utilizados:

```
<processModel maxWorkerThreads="20" maxIoThreads="20">
```

O parâmetro `maxWorkerThreads` e o parâmetro `maxIoThreads` são multiplicados implicitamente pelo número de CPU's. Por exemplo, se existirem dois processadores, o número máximo de *threads* de trabalho é o seguinte:

```
2*maxWorkerThreads
```

minFreeThreads e *minLocalRequestFreeThreads*

O ASP.NET também contém as seguintes definições de configuração que determinam quantos *threads* de trabalho e *threads* de portas de conclusão têm de estar disponíveis para iniciar um pedido remoto ou um pedido local:

```
<httpRuntime minFreeThreads="8"
minLocalRequestFreeThreads="8">
```

Se não existirem *threads* suficientes disponíveis, o pedido é colocado em fila de espera até que sejam disponibilizados threads suficientes para efectuarem o pedido. Por conseguinte, o ASP.NET não executará mais do que o seguinte número de pedidos ao mesmo tempo:

```
(maxWorkerThreads*número de CPUs)-minFreeThreads
```

Nota: Os parâmetros `minFreeThreads` e `minLocalRequestFreeThreads` não são multiplicados implicitamente pelo número de CPU's.

`minWorkerThreads`

Como o ASP.NET 1.0 Service Pack 3 e o ASP.NET 1.1, o ASP.NET também contêm a seguinte definição de configuração que determina quantos *threads* de trabalho poderão ser disponibilizados imediatamente para executar um pedido remoto.

```
<processModel minWorkerThreads="1">
```

As *threads* que são controladas por esta definição podem ser criadas mais rapidamente do que as *threads* de trabalho que são criadas a partir de funcionalidades predefinidas de "ajuste de *threads*" do CLR (*Common Language Runtime*). Esta definição permite que o ASP.NET execute pedidos que subitamente poderão estar a preencher a fila de pedidos ASP.NET em espera devido a um abrandamento num servidor *backend*, um súbito aumento de pedidos do lado do cliente, ou uma situação semelhante que poderá causar um aumento inesperado do número de pedidos em fila de espera. O valor predefinido do parâmetro *`minWorkerThreads`* é 1. A Microsoft recomenda que se defina o valor do parâmetro *`minWorkerThreads`* com o seguinte valor:

```
minWorkerThreads = maxWorkerThreads / 2
```

Por predefinição, o parâmetro `minWorkerThreads` não existe no ficheiro Web.config nem no ficheiro Machine.config. Esta definição é multiplicada implicitamente pelo número de CPU's.

`maxconnection`

O parâmetro `maxconnection` determina quantas ligações podem ser efectuadas para um endereço IP específico. O parâmetro é apresentado da seguinte forma:

```
<connectionManagement>
    <add address="*" maxconnection="2">
    <add address="65.53.32.230" maxconnection="12">
</connectionManagement>
```

As definições para os parâmetros abordados anteriormente nesta secção encontram-se todas em processamento. No entanto, a definição do parâmetro `maxconnection` aplica-se ao nível de *AppDomain*. Por predefinição, uma vez que esta definição se aplica ao nível de *AppDomain*, pode criar duas ligações no máximo para um endereço IP específico a partir de cada *AppDomain* no processo.

`executionTimeout`

O ASP.NET utiliza a seguinte definição de configuração para limitar o tempo de execução do pedido:

```
<httpRuntime executionTimeout="90"/>
```

Também pode definir este limite utilizando a propriedade `Server.ScriptTimeout` [14].

Nota: Caso se aumente o valor do parâmetro *`executionTimeout`*, também poderá ser necessário modificar a definição do parâmetro *`processModel responseDeadlockInterval`*.

4.5.1. Recomendações

As definições recomendadas nesta secção poderão não funcionar em todas as aplicações. No entanto, as informações adicionais que se seguem poderão ajudar a efectuar os ajustes adequados.

Se se estiver a efectuar uma chamada de Serviço Web para um único endereço IP a partir de

cada página ASPX, a Microsoft recomenda que se utilizem as seguintes definições de configuração:

- Definir os valores do parâmetro `maxWorkerThreads` e do parâmetro `maxIoThreads` como 100.
- Definir o valor do parâmetro `maxconnection` como **12*N** (em que N corresponde ao número de CPU's existentes).
- Definir os valores do parâmetro `minFreeThreads` como **88*_N_** e do parâmetro `minLocalRequestFreeThreads` como **76*_N_**.
- Definir o valor de `minWorkerThreads` como **50**. Notar que, `minWorkerThreads` não se encontra no ficheiro de configuração por predefinição, tendo de ser aicionado.

Algumas destas recomendações implicam uma fórmula simples que envolve o número de CPU's de um servidor. A variável que representa o número de CPU's nas fórmulas é *N*. Nestas definições, se se tiver a *hyperthreading* activada, terá de utilizar o número de CPU's lógicas em vez de o número de CPU's físicas. Por exemplo, se se tiver um servidor com quatro processadores com a *hyperthreading* activada, o valor de *N* nas fórmulas será 8 em vez de 4.

Quando se utiliza esta configuração, podem executar-se no máximo 12 pedidos ASP.NET por CPU ao mesmo tempo porque **100-88=12**. Por conseguinte, pelo menos **88*_N_** threads de trabalho e **88*_N_** *threads* de porta de conclusão estão disponíveis para outras utilizações (como para chamadas de retorno de Serviços Web).

Por exemplo, considerar que se tem um servidor com quatro processadores e *hyperthreading* activada. Com base nestas fórmulas, utilizam-se os seguintes valores:

```
<processModel maxWorkerThreads="100" maxIoThreads="100"
minWorkerThreads="50">
<httpRuntime minFreeThreads="704"
minLocalRequestFreeThreads="608">
<connectionManagement>
<add address="[ProvideIPHere]" maxconnection="96"/>
```

```
</connectionManagement>
```

Do mesmo modo, quando se utiliza esta configuração, existem 12 ligações disponíveis para CPU por endereço IP para cada *AppDomain*. Por conseguinte, no seguinte cenário, ocorre muito pouca contenção quando os pedidos aguardam por ligações e o *ThreadPool* não é totalmente utilizado:

- A Web hospeda apenas uma aplicação (*AppDomain*).
- Cada pedido para uma página ASPX corresponde a um pedido de Serviço Web.
- Todos os pedidos são para o mesmo endereço IP.

No entanto, quando se utiliza esta configuração, os cenários que envolvem uma das seguintes situações provavelmente utilizarão demasiadas ligações:

- Os pedidos são para múltiplos endereços IP.
- Os pedidos são redireccionados (código de estado 302).
- Os pedidos requerem autenticação.
- Os pedidos são efectuados a partir de múltiplos *AppDomains*.

Nestes cenários, deve ser utilizado um valor inferior para o parâmetro `maxconnection` e valores mais elevados para os parâmetros `minFreeThreads` e `minLocalRequestFreeThreads`.

4.5.2. Conclusões

Esta secção teve como objectivo, demostrar um conjunto de princípios de design, padrões e recomendações para permitir a criação de aplicações capazes de satisfazer os objectivos de desempenho, escalabilidade e performance em diferentes plataformas.

Projectar e desenvolver Serviços Web orientados ao desempenho e à performance tem vantagens e desvantagens. Em termos genéricos, o desempenho de um Serviço Web depende directamente da complexidade que lhe foi incutida, da velocidade da rede onde opera e da capacidade de processamento da máquina servidora. Atributos como, qualidade do serviço,

disponibilidade, gestão, integridade e segurança, também devem ser considerados e equilibrados de acordo com os objectivos de desempenho.

Modelar um processo de testes de desempenho, permite estabelecer objectivos por forma a optimizar tempo e recursos (CPU, memória, disco I/ O, e uma rede de I/ O). Devem ser usadas categorias, definidas pelo levantamento das necessidades dos testes de desempenho para a análise da aplicação e a abordagem de cada área, de modo a priorizar os problemas de desempenho. As categorias representam áreas-chave que frequentemente afectam o desempenho da aplicação, como estruturas de dados e algoritmos, comunicação, concorrência, gestão de recursos, de acoplamento e coesão, cache e gestão de estado.

5. Avaliação de Desempenho da Especificação de Segurança

Como se viu a utilização das especificações normativas apresentadas antes é feita através da inclusão de mais informação da definição dos serviços Web e como tal, é necessário estudar o impacto que esse acréscimo pode ter no desempenho dos serviços Web. Este capítulo apresenta um estudo e resultados do mesmo publicados em [16]. Após a apresentação do estudo é feita uma análise aos resultados, por forma a retirar ilações. A apresentação desse estudo também tem como objectivo mostrar a forma como se poderiam realizar estudos semelhantes para análise quer do desempenho quer de outros aspectos dos serviços Web.

A especificação *Web Service Security* (WSS), aprovada como um padrão pela OASIS e fortemente adoptada no mercado como uma solução para melhorar a segurança dos Serviços Web, continua a ser uma preocupação, devido ao conteúdo extra de segurança acrescentado à mensagem SOAP e do tempo suplementar para o processamento desse conteúdo adicional.

É do conhecimento comum que a especificação *WS-Security* adiciona uma sobrecarga significativa ao serviço de mensagens XML dos Serviços Web devido ao peso das operações de encriptação. Vários estudos [35] concluem que a especificação *WS-Security* atinge uma sobrecarga elevada e diminui a capacidade de processamento das mensagens. Os valores da sobrecarga situam-se em cerca de 50% quando as mensagens são assinadas ou encriptadas e cerca de 80% quando ambas são assinadas e encriptadas.

O estudo referido acima foi realizado por Kezhe Tang, Shiping Chen, David Levy, John Zic e Bo Yan ("*A Performance Evaluation of Web Services Security"*), com a finalidade de avaliar o desempenho de um Serviço Web tendo em conta a carga adicional que a Segurança impõe.

5.1. Objectivo do Teste

A utilização da norma WSS pode aumentar o tamanho de uma mensagem SOAP, o que pode levar ao estrangulamento e sobrecarga tanto do processamento da mensagem como na transmissão através da rede. A avaliação da sobrecarga será definida ao comparar a latência

de um Serviço Web com e sem definições WSS com uma gama de diferentes tipos de mensagem/ tamanhos.

O nível de segurança SOAP envolve intensas operações de computação de encriptação e de alocação de memória aquando do processamento das mensagens XML. Desta forma, torna-se importante compreender a performance da segurança SOAP em termos de dependências quer na implementação da encriptação quer nas propriedades dos documentos, cujo enfoque se situa na complexidade da estrutura [16, 35].

5.2. Mensagens

Ao implementar Encriptação XML e Assinatura Digital (Digital Signature) XML, juntamente com *tokens* de segurança, o WSS mantém as partes sensíveis da mensagem confidenciais e garante a integridade da mensagem enquanto a mensagem está em transacção. A figura 14 simula uma mensagem SOAP *"CustomerService"*, enquanto a mensagem SOAP na figura 15 ilustra o mesmo Serviço Web, mas desenvolvido com a política de encriptação WSS [16].

```
<soap:Envelope>
<soap:Header>
</soap:Header>
<soap:Body>
<createWorkOrdersResponse>
<createWorkOrdersResult>
<WorkOrder>
<customerID>1</customerID>
<customerName>Tang</customerName>
<addressStreet>A Street</addressStreet>
<addressCity>Sydney</addressCity>
<addressState>NSW</addressState>
<addressZip>2006</addressZip>
<sourceCompany>EE</sourceCompany>
<appointmentDate>210406</appointmentDate>
</WorkOrder>
</createWorkOrdersResult>
</createWorkOrdersResponse>
```

```
</soap:Body>
</soap:Envelope>
```

Figura 14 - Mensagem SOAP sem WSS [16]

```
<soap:Envelope>
<soap:Header>
<wsse:Security>
<wsse:BinarySecurityToken ValueType="...”>
MIICnzCCAgigAwIBAgIQBHB1ZCwolDXbdsxTrNLj
AMAsGA1UECxMEQ2VydDEMMAoGA1...
</wsse:BinarySecurityToken>
<xenc:EncryptedKey>
<xenc:EncryptionMethod Algorithm="…" />
<KeyInfo>
<wsse:SecurityTokenReference>
<wsse:Reference URI="… " …/>
</wsse:SecurityTokenReference>
</KeyInfo>
<xenc:CipherData>
<xenc:CipherValue>
iHlscgQVO4uCwztyCBwFzH8CIekMAoG
A1QBHB1gGjHa2GAKiaTaAgU…
</xenc:CipherValue>
</xenc:CipherData>
<xenc:ReferenceList>
<xenc:DataReference URI="…" />
</xenc:ReferenceList>
</xenc:EncryptedKey>
</wsse:Security>
</soap:Header>
<soap:Body>
<xenc:EncryptedData Id="…" …>
<xenc:EncryptionMethod Algorithm="…" />
<xenc:CipherData>
```

```
<xenc:CipherValue>
QtzfuLYO/qh45yDxaypPhI/YdH4bJ…
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</soap:Body>
</soap:Envelope>
```

Figura 15 - Mensagem SOAP com Encriptação WSS [16]

Analisando a figura 15, é possivel ver que foram adicionadas ao *header* SOAP duas etiquetas <wsse: Security>, em que os elementos de segurança exigidos foram adicionados. O elemento <wsse: BinarySecurityToken> contém as informações binárias do *token* de segurança (por exemplo, certificado X.509) utilizadas para a encriptação e o elemento <xenc: EncryptedKey> contém a chave para a encriptação. Dentro do elemento <xenc: EncryptedKey>, o elemento <xenc: EncryptionMethod> especifica o algoritmo de encriptação aplicada à encriptação e o elemento <KeyInfo> mantém uma referência ao elemento <wsse: BinarySecurityToken>. Uma vez que a chave de encriptação da mensagem pode ser encriptada, assim como o conteúdo do *body* SOAP, <xenc: CipherData>, no <xenc: EncryptedKey> o *header* fornece os dados encriptados da chave de encriptação, e o <xenc: EncryptedData> no *body*, contém os dados encriptados do conteúdo no *body*. A <xenc: ReferenceList> é usada como uma lista de referência para os dados encriptados do conteúdo no *body*. Também pode ser visto que o elemento <wsse: Security> e seus descendentes na mensagem encriptada tornam a mensagem SOAP muito maior do que a mensagem original.

Para a autenticação da mensagem encriptada, quando um envelope SOAP que contém os elementos de encriptação do *header* é recebido, qualquer chave de descodificação que está na posse do destinatário é identificada e os itens <xenc:EncryptedData> localizados (usando o <xenc:ReferenceList>) são decifrados pela chave de descodificação.

A Assinatura WSS funciona da mesma forma como a Encriptação WSS excepto que, apenas adiciona informações de assinatura no *header* SOAP e deixa o *body* inalterado. Esta assinatura é utilizada para verificar a mensagem assinada no *header*.

5.3. Metodologia

Foi utilizado um simples pedido/ resposta ao estilo RPC, modelo apresentado na figura 3, como cenário de teste, que consiste num cliente e num Serviço Web. O cliente envia um pedido para o Serviço Web de uma lista de registos de clientes. O Serviço Web recebe o pedido e responde enviando uma determinada quantidade de registos de clientes com ou sem aplicação de acordo com a WSS. Para evitar os efeitos colaterais de processos não relacionados com a medição de desempenho, por exemplo, acesso à base de dados e/ ou lógica de negócios, o Serviço Web apenas gera uma lista de objectos de dados e envia de volta para o cliente como uma resposta SOAP. O tempo total gasto foi medido no pedido e na resposta na máquina do cliente conhecido como latência.

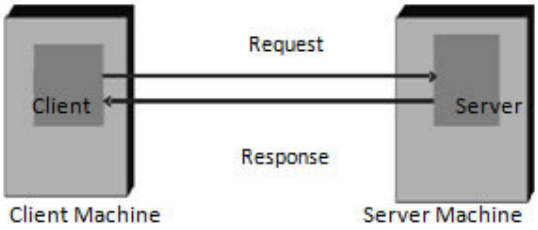

Figura 16 - Modelo RPC (pedido/ resposta) [16]

A API do Serviço Web pode ser definida em C # da seguinte forma:

```
public Customer[] GetCustomers(Request doc);
```

where:

```
public class Request
{
  int numCustomerRecords;
  bool isWSS;
  bool isEncryption;
  bool isSignature;
  bool isEncryptionSignature;
  bool isX509;
};
```

```
public class Customer
{
  public int customerID;
  public string customerName;
  public string addressStreet;
  public string addressCity;
  public string addressState;
  public string addressZip;
  public string sourceCompany;
  public DateTime appointmentDate;
};
```

O objecto de dados, o *Customer*, tem por objectivo reflectir uma estrutura de dados típica em aplicações e-business quer em termos de tamanho quer em complexidade. A classe *Customer* inclui algumas propriedades comuns para os clientes em aplicações reais. Os valores dos atributos de um objecto do *Customer* são providos por um gerador aleatório em tempo real, para evitar a cache ao nível do SO. Os quatro parâmetros no pedido especificam o tamanho da matriz do *Customer*, que permitem definir que configurações WSS se aplicam (encriptação, assinatura ou ambos), e que tipo de chave (chave simétrica ou assimétrica) se usa. Estes parâmetros podem ser especificados num ficheiro de configuração no terminal cliente de cada teste. Por exemplo, para configurar um teste que retorna uma matriz de 100 clientes com implementação de encriptação WSS usando o certificado X509, podemos simplesmente especificar o ficheiro de configuração abaixo:

```
<?xml version="1.0" encoding="utf-8" ?>
<TestConfig>
<NumberOfCustomers>100</NumberOfCustomers>
<WSSDeployed>true</WSSDeployed>
<SecurityOption>Encryption</SecurityOption>
<KeyType>X509</ KeyType >
</TestConfig>
```

5.4. Ambiente de Teste

Tanto o cliente como o Serviço Web foram desenvolvidos usando a Plataforma Microsoft .NET 1.1. O produto utilizado para a implementação foi o *WSS Enhancements* para Microsoft .NET (WSE) 2.0 SP3, que suporta *Web Services Security*. Todas as configurações do WSS são incorporadas no Serviço Web usando API´s WSE para suportar a configuração.

O Serviço Web e o Cliente são implementados e executados em duas máquinas ligadas através de uma rede LAN, cujas especificações são:

- Máquina 1 - Dell (Serviço Web):
    - CPU: Intel Core 2 Duo, 2.2GHz
    - Memória: 2.00GB
- Máquina 2 - Dell (Serviço Cliente):
    - CPU: Intel Core 2 Duo, 2.2GHz
    - Memória: 2.00GB

5.5. Ensaios

O objectivo deste ponto é perceber como uma política WSS afecta o desempenho dos Serviços Web. Os ensaios foram definidos e categorizados da seguinte forma:

- Definições WSS:
    - Não WSS
    - Encriptação WSS
    - Assinatura WSS
    - Encriptação + Assinatura WSS: Encriptação de uma mensagem assinada.
- Chave Pública utilizada para os testes
    - Nome de Utilizador
    - X509 Certificate

Foi utilizado o Nome de Utilizador e X509 como os exemplos de infra-estruturas simétricas e assimétricas de chaves públicas de encriptação e assinatura, sendo que, ambas são as principais e mais populares suportadas nos WSS.

Os ensaios foram realizados com mensagens de diferentes tamanhos, isto é, números de registos de clientes = 10, 100, 1000, para testar como WSS escala com diferentes tamanhos de mensagens.

A latência foi medida e comparada com e sem as configurações WSS. Foi definida a latência percentual de incremento (LIP) como uma métrica para avaliar a sobrecarga de desempenho para uma implementação específica WSS:

$$LIP = \frac{Lwssdeployment - Lnonwss}{Lnonwss} x100\%$$

onde:

L WSSDeployment é a latência do Serviço Web com um tipo específico de implementação WSS, por exemplo, *L WSSEncyption* para encriptação.

L NonWSS é a latência do Serviço Web sem qualquer implementação WSS.

5.6. Resultados dos testes e análise

Este parágrafo descreve os resultados obtidos no ensaio baseado na referência [16], sendo que foram adicionados outros dados para as conclusões e resultados provindos de outros estudos [18, 21, 35]. A Tabela 5 mostra os resultados dos ensaios. O tamanho de cada mensagem (resposta SOAP) é fornecido na Tabela 6 mapeando cada teste na Tabela 5 para o correspondente número de registos de clientes.

Número de Clientes		10	100	1000
Não WSS		18	35	227
WSS				
Encriptação	Nome de Utilizador	24	53	364
	X509	33	64	391
Assinatura	Nome de Utilizador	35	83	648

	X509	45	94	655
Encriptação + Assinatura	Nome de Utilizador	37	97	811
	X509	49	111	839

Tabela 6 - Latência (milissegundos)

Número de Clientes		10	100	1000
Não WSS		4527	34858	339059
WSS				
Encriptação	Nome de Utilizador	7048	47496	453084
	X509	8243	48691	454279
Assinatura	Nome de Utilizador	7583	37914	342115
	X509	8560	38891	343092
Encriptação + Assinatura	Nome de Utilizador	9609	50057	455645
	X509	10936	51384	456972

Tabela 7 - Tamanho da mensagem (bytes)

Encriptação WSS vs Não WSS

Conforme mostrado na Tabela 5, quando o Serviço Web com Encriptação WSS é solicitado para uma matriz de 100 clientes, a latência aumenta em cerca de 40% quando comparada com a latência da implementação do Não WSS. E quando a transacção aumenta para 1000 clientes, o LIP cresce aproximadamente 70%. Como resultado, podemos concluir que a queda de performance para Encriptação WSS está entre 30% e 70% para os tamanhos de mensagem especificados.

Assinatura WSS vs Não WSS

Inicialmente, assinando a mensagem com 100 clientes a latência sobe cerca de 130%, quando comparado com o teste sem o WSS. A tendência, conforme ilustrado cresce quase 200% quando o número de clientes chega a 1000. Comparando a Encriptação WSS e a Assinatura WSS (assinatura e verificação), a Assinatura WSS é muito mais pesada do que a Encriptação WSS (encriptação e descodificação).

Para melhor compreender este comportamento do desempenho, as mensagens foram capturadas na transacção, tanto para Encriptação WSS e testes de Assinatura. Como podemos verificar na Tabela 6, os tamanhos das mensagens encriptadas são maiores do que as mensagens assinadas. Isto significa que o tempo gasto para a mensagem com Assinatura WSS viajar através da rede, não deve ultrapassar o da Encriptação WSS. Assim, pode-se concluir que a assinatura/ verificação de uma mensagem SOAP custa mais ciclos de CPU do que a encriptação/ desencriptação.

Enquanto WSS fornece a capacidade de assinatura em mensagens SOAP, a assinatura padrão WSE é configurada para assinar, não só os dados da aplicação no *body* SOAP, mas também a hora e informações *Addressing* do Serviço Web no *header* SOAP. Por outro lado, a Encriptação .NET está configurada para encriptar somente o *body*. Esta pode ser a razão pelo que assinatura de uma mensagem SOAP leva mais tempo do que encriptação.

Relativamente aos tamanhos de mensagens assinadas, observa-se que as diferenças de tamanho entre uma mensagem assinada e a mensagem original são as mesmos que o número de alterações de registos de clientes. Isto porque, quando uma mensagem SOAP é assinada, apenas a informação relacionada com a assinatura é adicionada ao *header*, mas o conteúdo do *body* não muda. Assim, o tempo para uma mensagem assinada viajar é quase o mesmo que a transmissão da mensagem original com um pouco de tempo adicional para a transmissão de informação relacionada com a assinatura no *header*.

Encriptação + Assinatura WSS vs Não WSS

Em aplicações reais, Encriptação e Assinatura são normalmente utilizados para garantir a confidencialidade e a integridade dos dados transmitidos. Portanto, os testes foram realizados sobre o Serviço Web com uma combinação de implementação da Encriptação e Assinatura WSS, ou seja, a mensagem SOAP é assinada e encriptada antes de ser enviada.

Há um peso considerável no desempenho quando o Serviço Web é garantido por Encriptação e Assinatura WSS. O aumento é aproximadamente igual à soma do peso da encriptação e do peso da assinatura.

Nome de Utilizador vs X509

Para identificar o impacto no desempenho dos tipos de chave de segurança, um conjunto de testes WSS usando duas chaves de segurança diferentes foi efectuado, ou seja, Nome de Utilizador e X509.

A diferença entre as latências com Nome de Utilizador e X509 não são significativas para todos os testes, variando de 5 a 13 milissegundos. Além disso, a tendência de diferenças não aumenta como o tamanho das mensagens. Isto indica que o uso de chaves diferentes é improvável que tenha um grande impacto sobre a escalabilidade de Serviços Web em termos de tamanho da mensagem.

Uma vez que a utilização de chaves assimétricas oferece maior segurança para encriptação e assinatura de mensagens, o seu peso no desempenho não é tão elevado como se pudesse pensar. Assim, é recomendado o uso de certificados X509, especialmente quando é necessária forte segurança [16].

5.7. Conclusões

Foi apresentado um estudo sobre o desempenho de segurança nos Serviços Web e retiradas conclusões do mesmo. Uma referência simples foi utilizada para testar o desempenho do Serviço Web contra uma variedade de tamanhos de mensagem com ou sem a aplicação de políticas base de segurança dos Serviços Web. Observou-se que o peso da Assinatura WSS é muito superior à da Encriptação WSS. Observou-se, também, que a sobrecarga de desempenho de Encriptação e Assinatura WSS tendem a ser a soma de sobrecargas individuais. Além disso, usando Nome de Utilizador ou X509 em implementações WSS não cria diferenças significativa no desempenho.

A principal conclusão retirada é que a segurança ao nível das mensagens permite maior protecção do que a segurança ao nível do transporte, e é mais rápida. A título de curiosidade, para encriptar um *array* de 100KB, o processo demora cerca de 10 milissegundos, mas demora cerca de 100/ 200 milissegundos para processar operações de segurança SOAP.

6. Conclusões e Trabalho Futuro

Neste capítulo são discutidos os objectivos atingidos e não atingidos, analisadas as principais conclusões do trabalho realizado e apresentadas possibilidades de trabalho futuro. Também se apresenta, em jeito de conclusão, as perspectivas de evolução dos serviços Web.

6.1. Objectivos Atingidos e Conclusões do Trabalho

Em termos de objectivos atingidos, assumo que os objectivos definidos no início desta Tese foram, de uma forma geral, atingidos, exceptuando-se a componente prática de análise do desempenho dos serviços Web que por motivos vários não foi possível realizar. Apesar disso, houve o cuidado de mostrar (no capítulo 5) através da apresentação de um estudo da autoria de outros autores como se poderia realizar tal estudo.

A análise realizada relativamente às características e funcionalidade dos Serviços Web, como serviços/ aplicações baseadas na Internet, permitem concluir que a sua principal característica é uma flexibilidade espantosa. Suportados por uma arquitectura simples e eficaz, os serviços Web são potencialmente agentes agregadores de serviços entre diferentes sistemas, e este software extremamente versátil tem realmente o potencial de abrir uma nova era, a era da interoperabilidade, permitindo a integração de plataformas heterogéneas.

As pesquisas e estudos efectuados pelo autor, permitiram descrever o actual estado dos Serviços Web, evolução ao longo dos anos, conceitos inerentes e capacidades evolutivas (Amazon). Com raízes que remontam aos anos 70, os Serviços Web não são um produto fechado, evoluem naturalmente, atingindo maiores cotas de mercado de forma exponencial. O uso de Serviços Web tem-se tornado cada vez mais frequente pelo facto da independência da plataforma de desenvolvimento e operação.

Graças aos esforços conjuntos dos membros da WS-I, a interoperabilidade dos Serviços Web é, cada vez, maior. Com o surgimento de ferramentas IDE mais eficazes e eficientes, as integrações tendem a melhorar. Reportando a interoperabilidade à minha situação profissional, assumo esta característica dos Serviços Web de extrema importância. Consegue-

se, assim, integrar 3 sistemas distintos numa solução online de identificação e comércio de componentes automóvel (transacções B2B e B2C).

Os esforços de evolução resultam da crescente definição das especificações normativas que os Serviços Web devem respeitar e utilizar com o intuito de se tornarem sistemas mais robustos, fiáveis e seguros. Cada norma está associada a uma especificação de forma a permitir a melhor qualidade de serviço. Estas especificações tornam, assim, os Serviços Web sistemas únicos de transmissão de dados.

Podemos concluir que a qualidade dos serviços torna-se um requisito importante das transacções *business-to-business* e *business-to-consumer*. As várias propriedades de QoS (disponibilidade, acessibilidade, integridade, desempenho, fiabilidade, conformidade, interoperabilidade e segurança) precisam de ser tratadas na implementação das aplicações. Estas propriedades tornam-se ainda mais complexas quando são adicionadas necessidades de recursos transaccionais nos Serviços Web. Desta forma, algumas das limitações de protocolos como HTTP e SOAP podem dificultar a implementação de QoS, no entanto, QoS oferece as melhores práticas, nomeadamente na utilização de tipos de esquemas XML, *namespaces* e interfaces de Serviços Web.

Todos estes esforços visam aumentar a interoperabilidade, no entanto, concluo que problemas no desenvolvimento e conflitos dos Serviços Web são difíceis de corrigir quando implementados em ambientes de grande complexidade. Assumo esta premissa fruto da minha experiência profissional aquando da tentativa de integração da Plataforma Microsoft Sharepoint e Serviços Web SAP.

Este trabalho analisou e demonstrou resultados de testes sobre implementações de Serviços Web com o objectivo de avaliar características como desempenho. Uma conclusão importante revela que o desempenho varia consoante o número de utilizadores pendurados proporcionalmente ao tamanho das mensagens transaccionadas. O desempenho fica, assim, ainda mais afectado, se juntarmos encriptação e assinatura digital, de forma a tornar as transacções mais seguras.

A Segurança é um ponto fundamental dos Serviços Web. Sem ela, os Serviços Web são aplicações vulneráveis e com baixos índices de confiança. A introdução de Segurança nos Serviços Web vem de acordo com o ponto anterior, o desempenho. De uma forma sucinta, a equação é exponencial, isto é, quanto maior a segurança nos Serviços Web (assinaturas

digitais, encriptação, *tokens*, etc), menor será o desempenho do serviço devido à sobrecarga adicional de processamento.

Em jeito de conclusão final, os Serviços Web são produtos exclusivamente baseados em protocolos abertos da Internet. Esta é a sua maior vantagem, mas também uma significativa desvantagem, pois a força que permite a interoperabilidade, vem com o preço da velocidade, que é seriamente afectada pela largura de banda. Os Serviços Web não são perfeitos, mas certamente representam uma categoria de agentes de software que todos procuramos.

6.2. Capacidade Evolutiva

Os Serviços Web procuram estabelecer e explorar um forte e importante elo de ligação entre TI e estratégia empresarial. O que torna a abordagem aos Serviços Web relevante é o inerente custo-eficácia e eficiência de uma tecnologia que não exige enorme reformulação do código existente, que é de relativamente fácil aplicação tanto para novos sistemas como para já existentes.

A tecnologia dos Serviços Web, apesar de robusta, torna-se num elemento crítico para soluções empresariais nas áreas de *Enterprise Application Integration* (EAI), *Business Process Integration* (BPI), *Business-to-Business* (B2B) e *Electronic Data Interchange* (EDI). A chave para essas novas soluções é a forma que os Serviços Web permitem ligações "*loose couplings*" e "*on the fly*".

Serviços Web são *loose coupling* porque os seus objectivos apontam para uma interacção que é vagamente definida. Por exemplo, numa interacção entre um Serviço Web A e um Serviço Web B, A e B podem ser baseados em diferentes plataformas; a única coisa que A e B precisariam "conhecer" um sobre o outro seriam as informações contidas nas suas descrições baseado em XML (WSDL).

Da mesma forma, *on the fly* ou *late binding* descreve como um Serviço Web acede aos componentes e completa a sua ligação/ troca de informações. Apesar de não ser a primeira aplicação a usar o tipo ligações *late* (DLL's, registos e algumas classes Java também têm componentes ligações *late*) os Serviços Web levam este modelo para um novo nível, já que podem ser projectados de modo que os seus principais componentes sejam acedidos no momento da execução.

Grandes alterações são esperadas no futuro dos Serviços Web, seja em termos de Arquitectura, Segurança, Performance, Transformação de Dados, Processos de Negócio, Controlo de Versões ou ainda, Camadas de Acesso aos Serviços.

Em termos de Arquitectura, assistiremos a uma alteração que passa pela adopção do método *loose couple* e uma orientação às mensagens em detrimento do actual *tight couple*. Actualmente e no que respeita a Segurança, a mesma é disponibilizada pelos *endpoints* mas num futuro próxima passará por *gateways* ou *firewalls*.

Questões de performance também serão alvo de alterações, onde teremos um balanceamento de carga dividido pelas instâncias dos *endpoints* contrapondo o actual estado em que a perfomance depende do servidor da aplicação.

No que se refere a transformações de dados serão, estas serão completadas com um motor de transformação de dados embutido na Rede.

Em termos de Processos de Negócio, e de acordo com o princípio do ponto anterior, esta característica será também completada com motor de processos.

O Controlo da Versão, que actualmente é verificado nos *endpoints*, passará a ser verificado com base num *message router* baseado no *requester*, no cabeçalho e no conteúdo.

Por fim, as questões do tópico Camadas de Acesso aos Serviços, que actualmente são disponibilizadas pela aplicação, serão completadas com um motor de Qualidade de Serviço.

A tabela abaixo mostra um resumo do presente e uma previsão do futuro caminho de algumas características importantes dos Serviços Web [36]:

	Serviços Web Actuais	**Serviços Web Futuros**
Arquitectura	*Tightly coupled*, orientado ao terminal	*Loosely coupled*, orientado à mensagem
Segurança	Disponíbilizado por cada terminal	Disponibilizado por *gateways* ou *firewalls*
Desempenho	Depende da aplicação	Balanceamento de carga através de terminais

	servidora	instanciados
Transformação de Dados	Terminal e aplicação	Motor de tranformação de dados anexado à rede
Processos de Negócio	Processos associados a uma única aplicação	Processos reutilizados alojados num motor de processos
Controlo de Versão	Tratados no terminal	Router da Mensagem encaminha com base no requerente, cabeçalho e conteúdo
Acordos de nível de Serviço	Disponibilizados pela aplicação	Implementados e monitorizados por um motor de qualidade de serviço (QoS)

Figura 17 - Previsão do futuro da evolução das características dos Serviços Web [36]

6.3. Trabalho Futuro

Este trabalho analisou e demonstrou resultados de testes sobre implementações de Serviços Web tendo por base um estudo realizado por outros autores, e apresentado no capítulo 5. No entanto, e como referido acima não foi realizado neste trabalho um estudo prático que permitisse confirmar os resultados obtidos por esses autores, ou que os alargasse a outras especificações normativas "WS*." Desse modo, pretende-se como trabalho futuro colmatar essas falhas através da realização de um estudo experimental primeiramente com o objectivo de verificar esses resultados e posteriormente com a possibilidade de alargar a análise a outras especificações normativas.

7. Referências

[1] http://www.w3.org/TR/ws-arch/ (acedido em Agosto de 2009)

[2] http://www.w3schools.com/webservices/ws_intro.asp (acedido em Fevereiro de 2009)

[3] http://www.readwriteweb.com/archives/web_30_when_web_sites_become_web_services.php (acedido em Agosto de 2009)

[4] http://jodi.ecs.soton.ac.uk/Articles/v01/i01/BernersLee/ (acedido em Fevereiro de 2009)

[5] http://www.informationweek.com/news/software/development/showArticle.jhtml?articleID=6506480 (acedido em Agosto de 2009)

[6] http://www.databaseanswers.org/web_services_history.htm (acedido em Fevereiro de 2009)

[7] http://searchdatacenter.techtarget.com/sDefinition/0,,sid80_gci213925,00.html (acedido em Fevereiro de 2009)

[8] http://www.information-management.com/infodirect/20020517/5199-1.html (Acedido em Agosto de 2009)

[9] Stojanovic, Z. e Dahanayake, A. (2004). *Service-Oriented Software System Engineering: Challenges and Practices.*

[10] http://www.ibm.com/developerworks/library/ws-quality.html (acedido em Outubro 2009)

[11] http://java.sun.com/developer/technicalArticles/WebServices/restful/ (acedido em Outubro de 2009)

[12] http://www.testingreflections.com/node/view/3472 (acedido em Fevereiro de 2010)

[13]http://publib.boulder.ibm.com/infocenter/rpthelp/v7r0m0/index.jsp?topic=/com.ibm.ratinal.test.lt.ws.doc/topics/cwsreqs.html (acedido em Dezembro de 2009)

[14] http://support.microsoft.com/?id=821268 (acedido em Outubro de 2009)

[15] http://www.stylusstudio.com/ws_tester.html (acedido em Dezembro de 2009)

[16] K. Tang, S. Chen, D. Levy, J. Zic e B. Yan, “A Performance Evaluation of Web Services Security”, Proceedings of the 10th IEEE International - Enterprise Distributed Object Computing Conference (EDOC'06), 2006

[17] A. Nunes, S. Chen e P. Greenfield, “An Evaluation of Contemporary Commercial SOAP Implementations”, In Proceeding of the 5th Australasian Workshop on Software and System Architectures, Greenfield, http://mercury.it.swin.edu.au/ctg/AWSA04/Papers/ng.pdf

[18] H. Liu, S. Pallickara, G. H. Liu, S. Pallickara, G. Fox, “Performance of Service Webs Security”, In Proceedings of the 13th Annual Conference 13 de Mardi Gras, Baton Rouge, Louisiana, USA, February 2005, http://grids.ucs.indiana.edu/ptliupages/publications/WSSPerf.pdf

[19] K. Chiu, M.Govindaraju and R. Bramley, “Investigating the Limits of SOAP Performance for Scientific Computing”, In Proceedings in the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11).

[20] http://www.extreme.indiana.edu/xgws/papers/soap-hpdc2002/soap-hpdc2002.pdf

[21] D. Davis and M. Parashar, “Latency Performance of SOAP Implementations”, In Proceedings in the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002, http://www.caip.rutgers.edu/TASSL/Papers/p2p-p2pws02-soap.pdf

[22] Security in a Service Webs World: A Proposed Architecture and Roadmap Version 1.0, from IBM and Microsoft, 2002

[23] http://www-128.ibm.com/developerworks/library/specification/ws-secmap/

[24] Service Webs Security: SOAP Message Security 1.0 (WS-Security 2004), OASIS Standard 200401, Março 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[25] Service Webs Security: X.509 Certificate Token Profile, OASIS Standard 200401, Março 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

[26] Service Webs Security: Username Token Profile 1.0, OASIS Standard 200.401, Março 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

[27] Service Webs Trust Language (WS-Trust), Fevereiro 2005, ftp://www6.software.ibm.com/software/developer /library/ws-trust.pdf

[28] Service Webs Secure Conversation Language (WS-SecureConversation), Fevereiro 2005, ftp://www6.software.ibm.com/software/developer /library/ws-secureconversation.pdf

[29] Microsoft Service Webs Enhancements (WSE) 2.0 and 3.0 for .NET, Microsoft Service Webs Enhancements (WSE) 2.0 e 3.0 para .NET, http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx

[30] XML and Service Webs Security, XML e Service Webs Security, http://java.sun.com/webservices/xwss/index.jsp

[31] XML Encryption Syntax and Processing XML Encryption Syntax and Processing, http://www.w3.org/TR/xmlenc-core/

[32] http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest (acedido em Fevereiro de 2010)

[33] http://java.sun.com/blueprints/guidelines/designing_webservices/html/introduction2.html (acedido em Fevereiro de 2010)

[34] http://ws-i.org (acedido em Fevereiro de 2010)

[35] R. van Engelen, W. Zhang , “Identifying Opportunities for Web Services Security Performance Optimizations”, Department of Computer Science, Florida State University, Tallahassee, FL32306.

[36] S. Simnani, “Web Services – What the future holds”, Wipro Technologies, Munich, Germany.

[37] X. Xie, F.Li, “A configurable Web Service Performance Testing Framework”, Beijing, China, 2008

[38] R. Macedo, V. Mesquita, A. Caetano, A. Vasconcelos, J. Tribolet, “Web Services com o tecnología de suporte a processos de negócio”, Centro de Engenharia Organizacional, INESC INOV, Instituto Superior Técnico, Lisboa

Instituto Superior de Engenharia do Porto ⊙ Rua Dr. António Bernardino de Almeida, 431 ⊙ 4200-072 Porto
Tel. +351 228 340 500 ⊙ Fax +351 228 321 159 ⊙ mail@isep.ipp.pt ⊙ www.isep.ipp.pt