

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mestrado em Tecnologia Multimédia

WSQL

UMA ARQUITECTURA DE SOFTWARE

BASEADA EM WEB SERVICES

José Joaquim Magalhães Moreira
Licenciado em Engenharia Informática
pelo Instituto Superior de Engenharia do Porto

Dissertação submetida para satisfação parcial dos requisitos
do grau de mestre em Tecnologia Multimédia

Dissertação realizada sob a supervisão
do Professor Doutor Eurico Manuel Carrapatoso,
do Departamento de Engenharia Electrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

Porto, Abril de 2005

*A pessoa que usa as ferramentas
de ontem em trabalhos de hoje
não estará nos negócios amanhã.*

Business Week, the McGraw-Hill Companies

Resumo

A história da humanidade está marcada por grandes mudanças, de tal modo significativas que acabam por traçar decisivamente o rumo da sua evolução. Na classificação de Alvin Toffler para os grandes momentos verdadeiramente marcantes da história da humanidade, actualmente, estamos na terceira vaga de mudança que se iniciou na segunda metade do século XX, assinalando o começo de uma nova sociedade, a sociedade de informação, promovendo grandes inovações nas organizações. A Internet, como grande impulsionadora dessa sociedade, nos últimos anos, tem vindo a aumentar a sua importância em todos os contextos da vida moderna.

Embora possa parecer estranho, a história mostra que tem sido mais fácil uma pessoa interagir com os sistemas computacionais do que fazer dois sistemas interagirem entre si. Este paradoxo acontece porque, por um lado, a grande competição entre as empresas de tecnologia tem impedido que se desenvolvam normas de comunicação e, por outro lado, dada a inexistência de uma arquitectura de desenvolvimento que sirva como modelo para as aplicações interagirem.

Para ultrapassar estas limitações, desde o início do século XXI está a ser definida uma nova arquitectura, os Web Services, que aparece como um novo modelo para a computação distribuída, representando a última proposta de solução em desenvolvimento e integração de aplicações. Os Web Services surgem com a promessa de derrubar definitivamente a barreira de comunicação entre plataformas heterogéneas, revolucionando a forma de programar aplicações orientadas para a Internet, transformando-a numa arquitectura de *software* orientada a serviços.

Os Web Services são o tema central deste trabalho; assim, numa primeira parte, a tecnológica, apresentamos o estudo da arquitectura orientada a serviços e do conceito de Web Services, numa perspectiva de paradigmas computacionais e tecnológicos, descrevendo os protocolos e as normas. A segunda parte, descreve a actividade de desenvolvimento da aplicação wsQL - Web Services Query Language, que se concretiza como uma nova camada de *software*, tendo por objectivo receber comandos SQL de uma aplicação remota, e submete-los a uma base de dados, retornando o resultado para a aplicação requisitante. Foi definida uma arquitectura modular baseada em Web Services, que foi implementada na plataforma .NET. Seguiu-se a validação num cenário real de integração de aplicações e a submissão da aplicação ao teste de interoperabilidade segundo o perfil definido pelo WS-I, ficando desta forma demonstrado em pleno o potencial da aplicação wsQL.

Abstract

The History of Humanity is marked by profound changes, so significant that they end up tracing the route of its evolution. According to Alvin Toffler's classification for the truly important moments in our History, we are currently experiencing the third wave of changes initiated in the latter half of the 20th century, namely the beginning of a new information society, promoting great innovations in organizations. As the main drive of this new society, Internet has increased its influence in all contexts of modern life.

Strange as it may seem, History has shown us that it is easier for people to interact with computational systems, than to make two systems interact with each other. On the one hand, this paradox happens due to the great competition among technology companies, which will not allow communication norms to be developed, on the other hand, a development architecture that serves as a model for applications to interact is inexistent.

To overcome these limitations, a new architecture is under development since the beginning of the 21st century. Web Services come up as a new model for distributed computing and are the latest solution for applications development and integration. Web Services promise to bring down the communication barrier between heterogeneous platforms and revolutionize the way Internet-oriented applications are programmed, turning it into a service-oriented software architecture.

Web Services are the core subject of this paper, which begins by presenting a study of the service-oriented architecture and the concept of Web Services, in a perspective of computational and technological paradigms, describing the protocols and the norms. The second part, describes application development of wsQL - Web Services Query Language, materialized as a new layer of software, aiming to receive SQL commands from a remote application, submitting them to a database and delivering the result to the client-application. A model architecture based on Web Services has been defined and implemented using the .NET platform. Validation in a real context of application integration and the submission of the application to the interoperability test ensued, according to the profile defined by WS-I, confirming the full potential of the wsQL application.

Agradecimentos

Para concretizar esta missão que envolveu o estudo e a compreensão de alguns modelos de computação distribuída, de tecnologias antigas e de uma emergente que usam esses modelos e, posteriormente, organizar o conhecimento para elaborar uma dissertação, bem como usar esse conhecimento na criação de um protótipo capaz de demonstrar as capacidades da tecnologia emergente estudada, foram fundamentais um leque de apoios a vários níveis: ao nível emocional e familiar, ao nível de orientação, ao nível de trabalho e ao nível de condições necessárias para a experimentação do protótipo num cenário real.

Só porque se verificaram todas as condições especificadas é que foi possível criar o ambiente desejável para a criação do presente trabalho.

Embora a ordem usada não corresponda necessariamente à dos níveis referidos, começo, obviamente, por agradecer ao meu orientador, o Professor Doutor Eurico Manuel Carrapatoso, pela valiosa orientação, pela disponibilidade e dedicação demonstradas.

Um agradecimento especial é devido ao Professor Doutor António Godinho, o qual, na qualidade de Director Académico e Director do Curso de Sistemas de Informação e Multimédia do ISLA Gaia, proporcionou a flexibilidade do meu horário de aulas. Aproveito também, para agradecer à Direcção do grupo ISLA pela disponibilidade demonstrada para levar a cabo a experimentação do protótipo em todas as instituições do grupo.

Não posso deixar de agradecer aos meus pais, que estão na génese de todo o percurso para chegar até aqui e ao meu irmão pelas dicas. Em especial tenho que agradecer à Teresa, companheira e amiga, que se tornou minha mulher durante o cumprimento desta missão e, conforme prometido, me apoiou nos bons e nos maus momentos, suportou as minhas faltas de atenção para com ela e para com a nossa casa e que me foi ajudando dentro do que lhe era possível. Foi também ela a minha conselheira gráfica e correctora ortográfica. Por todo o amor, carinho, pela confiança em mim depositada, pela ajuda, motivação e pela companhia – Obrigado.

Índice

Resumo	V
Abstract.....	VII
Agradecimentos	IX
Índice	XI
Índice de figuras.....	XV
Lista de tabelas.....	XVII
Lista de exemplos	XVIII
Lista de acrónimos	XIX

CAPÍTULO 1

Introdução	1
1.1 Apresentação	1
1.2 Enquadramento	1
1.3 Objectivos	3
1.4 Organização da dissertação	5

CAPÍTULO 2

Arquitectura Orientada a Serviços	8
2.1 Contexto	8
2.2 A Arquitectura Orientada a Serviços	10
2.3 Exemplos de Arquitecturas Orientadas a Serviços	12
2.3.1 CORBA	13
2.3.2 DCOM	15
2.3.3 JavaRMI	16
2.3.4 Web Services	18
2.4 CORBA, DCOM e JavaRMI versus Web Services	19
2.5 Necessidade de uma nova arquitectura	21

CAPÍTULO 3

Web Services.....	23
3.1 Introdução	23
3.1.1 Problema: partilha de dados.....	25
3.1.2 A nova Internet	26
3.2 Definição de Web Service.....	28
3.3 Vantagens e desvantagens dos Web Services	31
3.4 Arquitectura dos Web Services.....	34
3.4.1 Modelo de implementação	36
3.4.2 Base protocolar	39
3.5 Protocolos básicos.....	41
3.5.1 HTTP.....	41
3.5.2 XML.....	42
3.5.3 SOAP	45
3.5.4 WSDL	54
3.5.5 UDDI.....	61
3.6 Protocolos avançados	64
3.6.1 Segurança.....	65
3.6.2 Outros protocolos avançados	73
3.7 Organizações que controlam as especificações.....	74
3.7.1 W3C	74
3.7.2 OASIS.....	75
3.7.3 WS-I.....	75
3.9 Construção de Web Services.....	77
3.9.1 Projecto de Web Services	77
3.9.2 Servidor.....	77
3.9.3 Cliente.....	78

CAPÍTULO 4

Especificação da Aplicação	80
4.1 Apresentação da aplicação wsQL	80
4.2 Objectivos da aplicação wsQL.....	81
4.3 Requisitos da aplicação	84
4.3.1 Requisitos funcionais	85

4.3.2	Requisitos não funcionais	85
4.3.3	Casos de utilização	86
4.4	Funcionamento lógico da aplicação wsQL	92
4.4.1	Servidor.....	92
4.4.2	Cliente.....	93
4.4.3	Diagramas de actividades	93
4.4.4	Diagramas de sequência	98
4.4.5	Etapas para a utilização	100

CAPÍTULO 5

Desenvolvimento da Aplicação		103
5.1	Introdução	103
5.1.1	Método de desenvolvimento	105
5.1.2	Plataforma de implementação	106
5.2	Subsistema do Web Service	107
5.2.1	Definição da arquitectura do sistema wsQL	108
5.2.2	Modelo de classes	110
5.3	Subsistema de Acesso a Dados	113
5.3.1	Modelo de classes	114
5.3.2	Construção da camada de dados	115
5.4	Subsistema de Gestão da Aplicação	118
5.4.1	Modelo de classes	118
5.4.2	Modelo relacional	120
5.4.3	Construção da aplicação de Gestão da Aplicação	121
5.5	Construção do Web Service wsQL	124
5.5.1	Operações do Web Service wsQL	124
5.5.2	Funções auxiliares do Web Service wsQL	128
5.6	Instalação do sistema wsQL	129
5.7	Validação do sistema wsQL	132
5.7.1	Cenário de validação do sistema wsQL	132
5.7.2	Aplicação cliente	133
5.7.3	Utilização da aplicação cliente	139
5.8	Avaliação de conformidade do wsOL	142

CAPÍTULO 6

Conclusão	145
6.1 Trabalho desenvolvido	146
6.2 Trabalho futuro.....	148
6.3 Considerações finais.....	149
Referências	151
Bibliografia	154

ANEXO A

Documento XML com esquema XML - XSD	160
---	-----

ANEXO B

Modelo de dados da Aplicação de Gestão	162
--	-----

ANEXO C

Código da Aplicação wsQL	164
C.1 Código do subsistema do Web Service	164
C.2 Componente Modulo.....	168
C.3 Código do subsistema de Acesso a Dados	173

ANEXO D

Documento WSDL do Web Service wsQL	183
--	-----

Índice de figuras

Figura 1.1 – Visão geral da aplicação	5
Figura 1.2 – Organização da tese	7
Figura 2.1 – Arquitectura Orientada a Serviços	11
Figura 2.2 – Arquitectura cliente-servidor de <i>stub/skeleton</i>	13
Figura 2.3 – Arquitectura do ORB do OMG	14
Figura 2.4 – Arquitectura DCOM	16
Figura 2.5 – Arquitectura JavaRMI	17
Figura 2.6 – Funcionamento lógico de um Web Services	18
Figura 3.1 – Cenário de utilização de Web Services	24
Figura 3.2 – Conceito de Web Service	29
Figura 3.3 – Invocação de procedimentos remotos	30
Figura 3.4 – Interface dos Web Services com sistemas de <i>back-end</i>	31
Figura 3.5 – Ciclo da interacção da aplicação cliente com o Web Service	34
Figura 3.6 – Interacção da aplicação cliente com o Web Service	36
Figura 3.7 – Modelo SOA [W3C, 2005b]	36
Figura 3.8 – Interacção entre as entidades do modelo SOA	37
Figura 3.9 – Principais fases da interacção entre as entidades do modelo SOA	38
Figura 3.10 – Pilha de protocolos dos Web Services [W3C, 2005b]	39
Figura 3.11 – Protocolos básicos dos Web Services	40
Figura 3.12 – Processo de transacção HTTP	41
Figura 3.13 – Estrutura geral de um documento XML	43
Figura 3.14 – SOAP visto em Camadas	46
Figura 3.15 – Envelope SOAP	47
Figura 3.16 – Conversão de dados: <i>string</i> <-> <i>Inteiro</i>	51
Figura 3.17 – Elementos do documento WSDL	55
Figura 3.18 – Arquitectura UDDI	62
Figura 3.19 – O registo UDDI baseia-se num Web Service	63
Figura 3.20 – Transporte seguro	66
Figura 3.21 – Comunicação segura com SSL	67
Figura 3.22 – Mensagem segura	67
Figura 3.23 – Criptografia de chave pública/privada	68
Figura 3.24 – Especificações associadas do WS-Security	71
Figura 3.25 – Protocolos avançados dos Web Services	73
Figura 3.26 – Um Web Service visto como três camadas	78

Figura 3.27 – Etapas de Descoberta, Descrição e Invocação de um Web Services	79
Figura 4.1 – Visão geral da aplicação wsQL	82
Figura 4.2 – A aplicação wsQL vista como uma nova camada de <i>software</i>	82
Figura 4.3 – Programação da interacção da aplicação cliente com a aplicação wsQL	83
Figura 4.4 – Diagrama de casos de utilização: gestor do sistema	87
Figura 4.5 – Diagrama de casos de utilização: aplicação cliente	89
Figura 4.6 – Diagrama de actividades: casos de uso do gestor sistema	94
Figura 4.7 – Diagrama de actividades: casos de uso da aplicação cliente	95
Figura 4.8 – Diagrama de actividades: casos de utilização <i>GetEstrutura</i> , <i>GetDados</i> e <i>ActualizaTabela</i>	96
Figura 4.9 – Diagrama de actividades: caso de utilização <i>ExecutaSQL</i>	97
Figura 4.10 – Diagrama de sequência: casos de uso do gestor do sistema	98
Figura 4.11 – Diagrama de sequência: casos de uso da aplicação cliente	99
Figura 4.12 – Aplicação wsQL no fornecedor dos serviços	101
Figura 4.13 – Programação da aplicação cliente	102
Figura 5.1 – Subsistemas do sistema wsQL	104
Figura 5.2 – O sistema wsQL visto em três camadas	107
Figura 5.3 – Arquitectura do sistema wsQL	109
Figura 5.4 – Diagrama de componentes do Web Service wsQL	109
Figura 5.5 – Modelo de classes do sistema wsQL	110
Figura 5.6 – Modelo de classes da camada de dados	114
Figura 5.7 – Identificação das classes	118
Figura 5.8 – Modelo de classes da aplicação de gestão do sistema	120
Figura 5.9 – Modelo relacional	120
Figura 5.10 – Diagrama do modelo relacional da aplicação de gestão do wsQL	121
Figura 5.11 – Interface gráfica da aplicação de gestão do sistema wsQL	122
Figura 5.12 – Diagrama de instalação da aplicação de gestão dos sistema wsQL	123
Figura 5.13 – Operações públicas do Web Service wsQL	125
Figura 5.14 – Desenvolvimento e instalação do Web Service wsQL	127
Figura 5.15 – Instalação dos ficheiros do sistema wsQL	130
Figura 5.16 – Diagrama de instalação do sistema wsQL	131
Figura 5.17 – Class proxy do wsQL	132
Figura 5.18 – Cenário para utilização do wsQL	134
Figura 5.19 – Aplicação cliente	135
Figura 5.20 – Camada de apresentação do Web Service	138
Figura 5.21 – Dados de uma tabela retornados pela operação <i>GetDadosTabela</i>	141
Figura 5.22 – Submeter instrução SQL ao sistema wsQL	142
Figura 5.23 - Resultado do teste de conformidade	144

Lista de tabelas

Tabela 2.1 – Comparação entre as várias arquitecturas	22
Tabela 3.1– Web tradicional versus Web Services	27
Tabela 3.2 – Comparação dos protocolos de segurança.....	72
Tabela 3.3 – Especificações dos Web Services	76
Tabela 5.1 – Operações definidas para cada uma das camadas	108
Tabela 5.2 – Descrição da classe wsQL.....	111
Tabela 5.3 – Descrição da classe Modulo.....	112
Tabela 5.4 – Descrição da classe AcessoBaseDados	115
Tabela 5.5 - Estrutura de campos definidos no esquema XML do exemplo 5.4.....	137
Tabela 5.6 – Estrutura de dados resultante do documento XML do exemplo 5.4	137
Tabela B.1 – Tabela Utilizadores.....	162
Tabela B.2 – Tabela PartilhasBds	162
Tabela B.3 – Tabela UtilizadoresPartilhas.....	163
Tabela B.4 – Tabela LigacoesOnLine.....	163
Tabela B.5 – Tabela LigacoesHistorico	163
Tabela B.6 – Tabela HistoricoOperacoes	163

Lista de exemplos

Exemplo 3.1 – Estrutura mensagem SOAP	47
Exemplo 3.2 – Elemento soap:Envelope numa mensagem SOAP	48
Exemplo 3.3 – Elemento soap:Header numa mensagem SOAP	49
Exemplo 3.4 – Elemento soap:Body numa mensagem SOAP	49
Exemplo 3.5 – Elemento soap:Fault numa mensagem SOAP	51
Exemplo 3.6 – Esquema XML numa mensagem SOAP	52
Exemplo 3.7 – Requisição SOAP embebida em HTTP	53
Exemplo 3.8 – Resposta SOAP embebida em HTTP	53
Exemplo 3.9 – Elemento <i>definitions</i> no documento WSDL	56
Exemplo 3.10 – Elemento <i>types</i> no documento WSDL	56
Exemplo 3.11 – Elemento <i>messages</i> no documento WSDL	57
Exemplo 3.12 – Elemento <i>operation</i> no documento WSDL	58
Exemplo 3.13 – Elemento <i>portType</i> no documento WSDL	59
Exemplo 3.14 – Elemento <i>binding</i> no documento WSDL	60
Exemplo 3.15 – Elemento <i>port</i> no documento WSDL	60
Exemplo 3.16 – Elemento <i>service</i> no documento WSDL	61
Exemplo 5.1 – Código do componente da camada de dados	116
Exemplo 5.2 – Exemplo de codificação de um Web Service	126
Exemplo 5.3 – Exemplo de codificação do modulo	128
Exemplo 5.4 – Lista de parâmetros retornada pelo Web Service	135
Exemplo 5.5 – Invocação das operações <i>GetChavePublica</i> e <i>Login</i>	140
Exemplo A.1 – Documento XML com esquema XML	160
Exemplo C.1.1 – Código do Web Service	164
Exemplo C.2.1 – Código do Modulo	168
Exemplo C.3.1 – Código da camada de dados	173
Exemplo D.1 – Documento WSDL do Web Service wsQL	183

Lista de acrónimos

A2A	<i>Aplication-to-Aplication</i>
API	<i>Aplication Programming Interface</i>
ASCII	<i>American Code for Information Interchange</i>
ASMX	<i>Active Server Method Extension</i>
ASP	<i>Active Server Pages</i>
B2B	<i>Business-to-Business</i>
B2C	<i>Business-to-Consumer</i>
CA	<i>Certificate Authority</i>
COM	<i>Component Object Model</i>
CORBA	<i>Common Object Request Broker Architecture</i>
CSS	<i>Cascading Style Sheets</i>
DES	<i>Data Encryption Standard</i>
DCE	<i>Distributed Computer Enviroment</i>
DCOM	<i>Distributed Component Object Model</i>
DLL	<i>Dynamic Link Libray</i>
DTD	<i>Document Type Definition</i>
EAI	<i>Enterprise Aplication Integration</i>
FTP	<i>File Transfer Protocol</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
IETF	<i>Internet Engineering Task Force</i>
IDL	<i>Interface Definition Language</i>
IOP	<i>Internet Inter-ORB Protocol</i>
IIS	<i>Internet Information Services</i>
IP	<i>Internet Protocol</i>
IPsec	<i>IP Security</i>
IRI	<i>Internationalized Resource Identifiers</i>
JavaRMI	<i>Java Remote Method Invocation</i>
JMS	<i>Java Message Service</i>

JSP	<i>Java Server Pages</i>
JVM	<i>Java Virtual Machine</i>
MIME	<i>Multipurpose Internet Mail Extensions</i>
MSMQ	<i>Microsoft Message Queue</i>
OMG	<i>Object Management Group</i>
OASIS	<i>Organization for the Advancement of Structured Information Systems</i>
RPC	<i>Remote Procedure Call</i>
S-HTTP	<i>Secure Hypertext Transfer Protocol</i>
SMTP	<i>Simple Mail Transfer Protocol</i>
SOAP	<i>Simple Object Access Protocol</i>
SSL	<i>Secure Sockets Layer</i>
SQL	<i>Structured Query Language</i>
TI	<i>Tecnologia de Informação</i>
TIC	<i>Tecnologia de Informação e Comunicação</i>
TCP	<i>Transmission Control Protocol</i>
TCP / IP	<i>Transmission Control Protocol / Internet Protocol</i>
UDDI	<i>Universal Description, Discovery and Integration</i>
UDP	<i>User Datagram Protocol</i>
UML	<i>Unified Modelling Language</i>
URI	<i>Uniform Resource Identifier</i>
URL	<i>Uniform Resource Locator</i>
W3C	<i>World Wide Web Consortium</i>
wsQL	<i>Web Service Query Language</i>
WWW	<i>World Wide Web</i>
Web	<i>World Wide Web</i>
WSA	<i>Web Services Architecture</i>
WSDL	<i>Web Services Description Language</i>
XML	<i>Extensible Markup Language</i>
XSD	<i>XML Schema Data</i>
XSL	<i>Extensible Stylesheet Language</i>
XSLT	<i>Extensible Stylesheet Language Transformations</i>

Introdução

1.1 Apresentação

“O homem passa o tempo a montar mecanismos de que se torna depois prisioneiro mais ou menos voluntário”. A frase, de Marc Bloch, historiador Francês, fuzilado pelos nazis em 1945, ajustada à paisagem rural do norte de França, é singularmente válida se a remetermos para o contexto da comunicação entre aplicações informáticas. Embora possa parecer estranho, tem sido mais fácil uma pessoa interagir com sistemas computacionais do que fazer dois sistemas interagirem entre si. Este paradoxo acontece porque, por um lado, a grande competição entre as empresas de tecnologia tem impedido que se desenvolvam normas comuns de comunicação e, por outro, dada a inexistência de uma arquitectura de desenvolvimento que sirva como modelo para as aplicações interagirem.

Assim, durante décadas foram sendo criadas ilhas de dados, no oceano dos sistemas de informação, centradas nas aplicações, que trabalhavam isoladamente; como resultado mais uma babel nasceu, desta feita computacional.

Para colmatar este problema, dos sistemas não “falarem a mesma língua”, exigia-se uma nova abordagem de desenvolvimento de aplicações: um modelo de Arquitectura Orientada a Serviços, e um conceito que suportasse este modelo, os Web Services.

1.2 Enquadramento

Uma sociedade em constante mudança coloca um permanente desafio às organizações. As tecnologias de informação e comunicação, TIC, das quais o computador

e a Internet são os dois expoentes máximos, são um dos factores mais salientes dessa mudança acelerada, a que as organizações têm de ser capazes de se antecipar, reagir rapidamente e mesmo promover.

As TIC têm funcionado como a principal força impulsionadora da revolução tecnológica a que actualmente assistimos. Como resultado, os sistemas computacionais distribuídos vivem momentos de grande agitação, que são indicativos de rápidas e significativas mudanças. Como elementos catalisadores desta dinâmica temos, por um lado, os avanços tecnológicos e, por outro, a competição entre fabricantes que lutam pela conquista de maiores fatias de mercado neste sector.

Associada a esta revolução, verifica-se também uma alteração dos hábitos de trabalho, lazer e mesmo de pensar. Hoje, os recursos humanos são considerados o recurso mais valioso das organizações; consequentemente o tempo que estes gastam na realização das suas tarefas tornou-se num factor preponderante na competitividade das organizações. Como tal, a evolução a que se tem assistido tem sido sempre no sentido de se arranjam formas de diminuir o tempo gasto na realização das tarefas, ou então de automatizar essas tarefas, especialmente as repetitivas.

No contexto do desenvolvimento de *software* essas formas têm passado por algumas abstracções tais como: objectos, modelo cliente/servidor, plataformas distribuídas, reutilização, portabilidade, integração, entre outros.

Como resultado, foram aparecendo várias arquitecturas de objectos distribuídos, das quais se destacam, pela sua importância no panorama actual dos sistemas de informação, as seguintes: CORBA, COM/DCOM e JavaRMI.

No entanto, algumas limitações de cada uma destas tecnologias impedem que sejam usadas em grande escala. Essas limitações resultam de causas como: falta de interoperabilidade, complexidade de implementação, centrada numa dada plataforma, altamente dependente do código e comunicação tipicamente síncrona.

Tim Berners-Lee, um dos responsáveis pelo conceito World Wide Web, vulgo WWW ou *Web*, definiu-o na criação como: “um espaço onde a informação tem um significado bem definido, facilitando a cooperação entre pessoas e agentes computacionais” [Berners-Lee, 2002]. Contudo, a história recente da *Web* demonstra que, das duas facilidades anunciadas, apenas a primeira foi conseguida na plenitude; da segunda, podemos dizer que estamos a caminho. Esta tese leva-nos na direcção desse destino.

Entretanto, o consórcio, denominado por W3C, principal criador de normas para a *Web*, definiu como missão elevar a *Web* ao seu expoente máximo. Como princípio de actuação, define a *Web* como um universo de informações acessíveis através de qualquer dispositivo ligado à Internet. Com isto, pretende que a informação possa estar disponível por meio de computadores, telefones, de televisores ou mesmo de outros dispositivos conectados à rede.

O princípio de universalidade de acesso, sem limitação de plataforma de *software* ou hardware, infra-estrutura de rede, idioma, cultura, localização geográfica e limitações físicas ou mentais, é um ponto-chave no projecto da *Web* do futuro.

Nos últimos anos, a Internet tem vindo a aumentar a sua importância e aplicabilidade na computação distribuída. A sua simplicidade e onnipresença tornam-na uma escolha lógica como espinha dorsal para aplicações distribuídas.

Como nos encontramos numa época de globalização crescente, as Tecnologias de Informação e Comunicação, e em especial a Internet, emergem para assumir uma posição privilegiada para a comunicação entre aplicações, com essencial ênfase na integração de tecnologias e sistemas computacionais.

Consequentemente, a próxima geração da *Web* será relativa a dados e não a texto, assentando numa arquitectura orientada a serviços de *software*, para fazer com que as aplicações possam interoperar directamente. A integração de aplicações atingirá o seu auge e essas aplicações serão construídas pela combinação de objectos, aplicações e bases de dados.

1.3 Objectivos

Integrar tecnologias e sistemas computacionais exige uma nova abordagem de desenvolvimento de *software*, que incorpore o conceito de arquitectura cooperativa na arquitectura de componentes dirigidos a serviços.

A Arquitectura Orientada a Serviços representa um novo modelo de desenvolvimento de aplicações, que constitui uma mudança para uma arquitectura de *software* orientada a serviços, relativamente distinta da arquitectura orientada a objectos, hoje comum.

Assim, segue um modelo de serviços, ou seja, preconiza que o *software* do futuro seja projectado como uma colecção de serviços em rede, que podem interagir uns com os outros, em que se privilegia o uso de normas abertas.

Os Web Services, também designados por serviços abertos ou serviços globais, são o mais recente e promissor modelo de computação distribuída baseada em serviços, representando a última proposta de solução para o desenvolvimento e integração de aplicações. Surgem em resposta aos problemas levantados durante décadas pela falta de interoperabilidade e integração de aplicações. Representam também uma viragem na forma de actuação das empresas da área, onde os Web Services estão na ordem do dia e de trabalho, que estão empenhadas, e finalmente a trabalhar juntas, no desenvolvimento de protocolos abertos para este novo paradigma, deixando a competição para outros planos.

Entretanto, por se tratar de tecnologia emergente, a definição de alguns aspectos ainda se encontra em aberto e em discussão, surgindo quase todos os dias novas especificações. No entanto, tudo aponta para um grande sucesso dos Web Services, que se deve especialmente à adopção de uma série de normas com grande sucesso na Internet, como HTTP, SOAP e XML.

Esta dissertação surge neste contexto de agitação ao nível dos sistemas de informação distribuídos e, portanto tem essencialmente dois objectivos. O primeiro é estudar a arquitectura de *software* orientada a serviços e as tecnologias mais representativas estabelecendo as limitações e as causas para a necessidade de uma nova tecnologia. Estudado o modelo teórico passaremos ao estudo do tema principal da dissertação, o conceito dos Web Services numa perspectiva de paradigmas computacionais e tecnológicos.

O segundo objectivo desta dissertação é a construção de aplicações seguindo um modelo de Arquitectura de *software* Orientada a Serviços. O nosso propósito principal é demonstrar as capacidades deste novo modelo e utilizar o expoente máximo deste paradigma, os Web Services, na elaboração de uma aplicação de *software*, que baptizámos com o nome: wsQL – Web Service Query Language, que tire partido de todos estes conceitos novos e emergentes.

A aplicação pretende-se que seja um Web Service que permita receber comandos SQL, *Structured Query Language*, de uma aplicação remota, e os submeta a uma determinada base de dados, retornando o resultado para a aplicação requisitante. Uma visão geral da aplicação pode ser vista na figura 1.1.

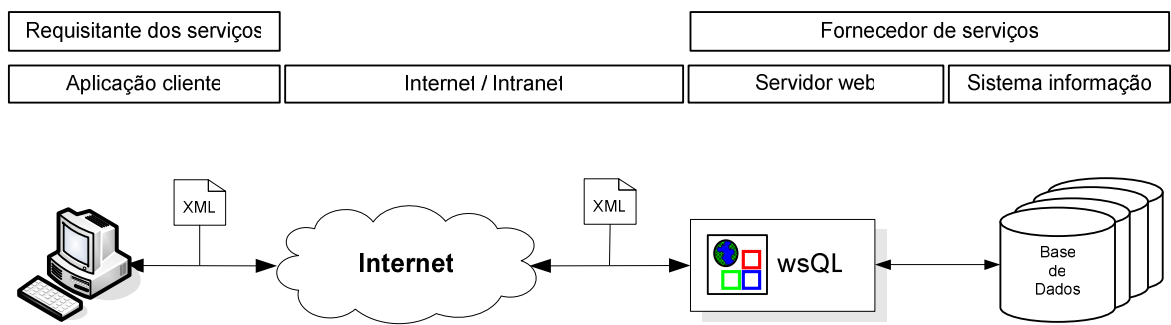


Figura 1.1 – Visão geral da aplicação

O objectivo principal deste Web Service é permitir a uma organização expor, via *Web*, um conjunto de funções, para serem acedidas programaticamente, que permitam receber comandos para interrogação ou manutenção de uma base de dados.

Pode ser considerado como uma nova camada de *software*, que uma organização coloca entre as suas bases de dados e os seus parceiros internos ou externos. Permitindo, assim a total integração de aplicações entre a organização, fornecedora dos serviços ou funções, e os seus parceiros, requisitantes do serviço.

Apesar da aplicação desenvolvida ser suficientemente flexível para a poder aplicar a qualquer contexto de partilha de bases dados para acesso remoto, vamos apresentar um estudo de caso real de integração de aplicações. A aplicação será usada no seguinte cenário: uma instituição de ensino superior com quatro pólos geograficamente distantes, Bragança, Gaia, Leiria e Santarém. Cada instituição poderá aceder à base de dados do sistema de informação escolar, de cada uma das três outras instituições.

1.4 Organização da dissertação

Neste primeiro capítulo, contextualizamos o tema da dissertação, ou seja, o problema a tratar que se insere na categoria dos sistemas de informação distribuídos, tendo como espinha dorsal a Internet. Apontamos também os objectivos e apresentamos a aplicação de suporte a esta tese, a aplicação wsQL.

O segundo capítulo aborda as principais características da Arquitectura Orientada a Serviços. São analisadas as tecnologias mais relevantes que tentaram seguir este modelo, é

efectuada uma análise comparativa, e são apontadas as limitações. Apresenta-se de seguida a tecnologia que melhor segue este modelo, os Web Services. É aqui que se estabelece a ponte com o capítulo terceiro, que é dedicado por inteiro aos Web Services.

Assim, o capítulo terceiro divide-se em duas partes. A primeira é dedicada às principais motivações que impulsionaram esta nova arquitectura, continua com a sua definição e fundamentos teóricos que a suportam. A segunda parte inicia-se com o estabelecimento da relação dos Web Services com a Arquitectura Orientada a Serviços, prossegue com o estudo dos protocolos que sustentam toda a arquitectura dos Web Services, agrupando-os em protocolos básicos e protocolos avançados, finalizando o capítulo com as principais linhas de orientação para construir aplicações baseadas em Web Services.

O capítulo quarto começa com a apresentação da aplicação de suporte a esta dissertação, destacando os principais objectivos. Seguidamente são descritos os requisitos funcionais e não funcionais e os casos de uso de cada um dos actores do sistema. Na parte final do capítulo são descritas as etapas para a utilização da aplicação, quer por parte da organização fornecedora dos serviços, quer por parte da aplicação cliente.

O capítulo quinto divide-se em duas partes: A primeira descreve toda a actividade de desenvolvimento do sistema wsQL que é constituído pelo subsistema Web Service wsQL, o subsistema do componente de acesso a bases de dados, o subsistema da aplicação de gestão do sistema e a aplicação cliente para testar o Web Service. A segunda, inicia-se com a descrição das etapas de instalação do sistema wsQL, continua com a avaliação do sistema num contexto real, sendo finalizado com o resultado da avaliação da conformidade do wsQL com o perfil de interoperabilidade definido pelo WS-I, *Web Service Interoperability Organization*.

Finalmente, com o sexto capítulo, apresenta-se as contribuições principais do trabalho, as linhas orientadoras que guiarão evoluções futuras do sistema wsQL e algumas sugestões para outras actividades de I&D relacionadas.

Na figura 1.2 podemos analisar um diagrama que ilustra a organização lógica desta dissertação.

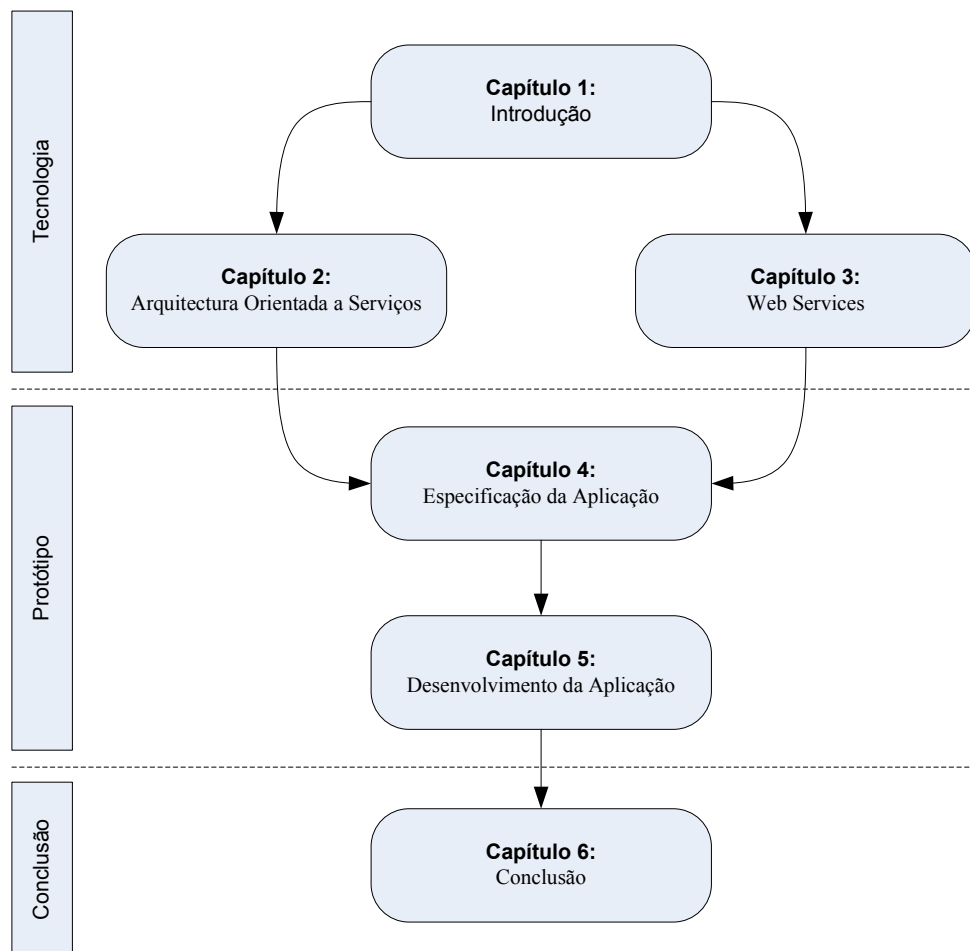


Figura 1.2 – Organização da tese

Arquitectura Orientada a Serviços

2.1 Contexto

A história da humanidade fica vincada por grandes mudanças, de tal modo significativas que acabam por traçar decisivamente o rumo da sua evolução. Alvin Toffler [Toffler, 1984] definiu três grandes momentos verdadeiramente marcantes a que chamou de vagas de mudança na história do Homem. Assim, definiu a revolução agrícola iniciada há cerca de dez mil anos como a primeira grande vaga de mudança, a revolução industrial do finais do século XVII como a segunda vaga e uma terceira vaga iniciada em meados do século XX, assinalando o começo de uma nova sociedade, a sociedade de informação, na qual a informação e o conhecimento assumem um papel fulcral.

Em Portugal, o Livro Verde para a Sociedade de Informação define esta sociedade como: “Modo de desenvolvimento social e económico em que a aquisição, armazenamento, processamento, valorização, transmissão, distribuição e disseminação de informação conducente à criação de conhecimento desempenham um papel central na actividade económica, na criação de riqueza, na definição da qualidade de vida dos cidadãos e das suas práticas culturais” [MSI, 2004].

Daqui se conclui que a evolução da sociedade actual não será motivada pela energia nem pela força, mas sim pelo domínio da informação.

No epicentro desta nova sociedade encontramos as Tecnologias de Informação e Comunicação, TIC. Estas, ao evoluírem tão rapidamente ao longo das últimas décadas, levam por arrasto a transformações nos mais diversos aspectos circundantes da nossa vida e na própria essência dos negócios.

Perante o grande potencial das TIC, as organizações adoptam-nas, quer através da optimização do seu funcionamento interno ou em comunicação com parceiros de negócio,

quer induzindo alterações ao nível do próprio negócio, tornando-as assim num dos instrumentos mais valiosos da gestão moderna.

Contudo, apesar da sua importância para a sobrevivência e evolução das organizações, a sua adopção não garante a obtenção de quaisquer vantagens competitivas, não havendo assim uma relação directa entre a sua adopção e o retorno esperado: este último depende do modo como são utilizadas [Li, 1995].

Entretanto, à medida que todas as organizações têm acesso à mesma base tecnológica, as vantagens competitivas daí decorrentes serão resultado de uma gestão e utilização criativa e de conseguir aproveitar da melhor forma o grande potencial da tecnologia.

É um facto que a complexidade das organizações está a aumentar, que resulta da crescente diversidade de novos conceitos e tecnologias. Por outro lado, encontramos-nos numa época de globalização crescente que tem vindo a modificar a vida de todos. Surgem novos modelos e sistemas de negócio, principalmente os desenvolvidos e suportados no âmbito da Internet, onde se torna vital a interoperação entre parceiros num contexto aberto e global. Conseguir estar em pleno neste contexto, tornar-se-á pois um factor de diferenciação e competitividade das organizações. Portanto, é requerido também um sistema de informação que corresponda a estas novas necessidades de informação.

Neste contexto, os sistemas de informação têm que englobar muito mais do que aplicações isoladas. Hardware de baixo custo, elevada capacidade de processamento e normas industriais tornaram-se parte importante e integrante dessas mesmas aplicações. Com esta mudança, a definição de aplicação distribuída, que tradicionalmente é definida como uma aplicação escrita numa linguagem de programação, qualquer que ela seja, mas que usualmente será baseada em objectos, e que se executa num ou mais computadores, utilizando as funcionalidades oferecidas pelos *softwares* de suporte a aplicações distribuídas, terá que evoluiu para incluir muito mais do que apenas *software*.

Dada a natureza destes sistemas distribuídos, e as interdependências associadas entre hardware e *software*, verificou-se um aumento significativo da complexidade de todo o ciclo de vida das Tecnologias de Informação, TI. Segundo a Microsoft Portugal, as grandes empresas gastam cerca de 70 a 80 por cento do orçamento TI na manutenção dos sistemas existentes devido à natureza manual de muitos processos de operações contínuas. Apesar de muitos destes problemas serem apenas sentidos pelas grandes empresas, as pequenas e médias empresas deparam-se também com os desafios da complexidade crescente das suas infra-estruturas de TI.

A resposta a estes desafios terá de passar por uma mudança de paradigma, ou seja, o *software* deverá mudar o foco de uma aplicação simples, para uma perspectiva que engloba todo o sistema distribuído. Esta mudança deve ser eficaz em áreas chave [Microsoft, 2005]:

- Os modelos de desenvolvimento devem evoluir para permitir a concepção de sistemas distribuídos completos, incluindo aplicações e os requisitos operacionais relacionados, tais como topologia, definições de segurança e recursos de hardware;
- Os sistemas operativos devem evoluir da gestão de recursos de informática, de armazenamento e de rede num único servidor, para a gestão de recursos distribuídos por vários centros de dados; os sistemas operativos devem ter a capacidade de permitir a implementação e operação eficiente e eficaz de sistemas distribuídos e disponibilizar uma visualização a nível de sistema de todos os recursos associados;
- As soluções de gestão devem evoluir para além de uma perspectiva de infraestrutura focada em partes individuais para oferecer uma perspectiva aprofundada centrada no sistema que abranja os recursos de hardware subjacentes para possibilitar uma administração automatizada, auto-contida e abstraída de questões específicas das aplicações.

2.2 A Arquitectura Orientada a Serviços

A Arquitectura Orientada a Serviços (do inglês *Service-Oriented Architecture* – SOA) [W3C, 2005b] preconiza um outro tipo de abordagem no desenvolvimento de *software*, no qual as aplicações passam a ser construídas e reorganizadas como fornecedoras de serviços ou operações específicos e bem definidos e que podem ser publicados.

Esta arquitectura é denominada “orientada a serviços” porque a base principal são os serviços, que um cliente, podendo ser uma pessoa ou uma aplicação, necessita que sejam realizados. Neste contexto, podemos definir um serviço como um processo de negócio, ou seja um conjunto de funcionalidades computacionais, reutilizável, que pode ser acedido através da *Web*.

Estas arquitecturas devem oferecer mecanismos de descrição e organização dos serviços, de descoberta dinâmica e de adaptação automática. Para que este modelo de arquitectura se torne exequível, o conjunto de serviços deve estar organizado em categorias hierárquicas e acessíveis, descrevendo o que cada serviço faz e como pode ser invocado. Estas categorias estarão disponíveis através de serviços definidos para este fim, disponibilizados por entidades intermediárias.

Basicamente, fazem parte desta arquitectura três elementos: um fornecedor de serviços, um invocador de serviços e um mediador de serviços:

- **Fornecedor de serviços** – o fornecedor de serviços contém um ou mais serviços que são disponibilizados aos invocadores de serviços, ou seja os clientes, e para viabilizar o acesso aos serviços por parte dos clientes, o fornecedor divulga os seus serviços nos repositórios do mediador de serviços;
- **Invocador de serviços** – o cliente, é o elemento que faz a solicitação dos serviços, portanto, é quem inicia a comunicação, podendo utilizar as funções dos serviços solicitados para a composição de outros serviços;
- **Mediador de serviços** – os mediadores de serviços armazenam a descrição, a classificação e localização dos serviços disponibilizados pelos fornecedores para todos os participantes do sistema. Os mediadores têm um papel fundamental no funcionamento da arquitectura, pois facilitam a descoberta dinâmica dos serviços por parte dos clientes.

A figura 2.1 mostra a interacção entre os três elementos e os papéis desempenhados nesta Arquitectura Orientada a Serviços.

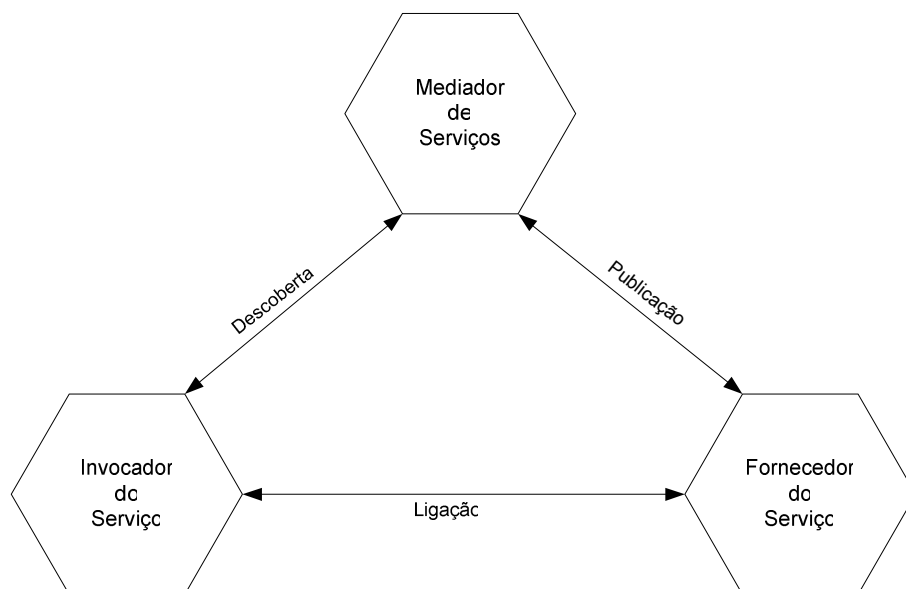


Figura 2.1 – Arquitectura Orientada a Serviços

As colaborações e interacções entre os três elementos são suportadas por um protocolo normalizado. Cada serviço é descrito num formato normalizado e essas descrições são um elemento chave para todos os três papéis, pois é delas que se extraem as informações necessárias para classificar, escolher e invocar um serviço.

A Arquitectura Orientada a Serviços é essencialmente uma colecção de serviços, que comunicam uns com os outros. A comunicação pode envolver a passagem simples de dados ou a coordenação da actividade de dois ou mais serviços [Douglas, 2003].

Contudo, os conceitos subjacentes a esta arquitectura não são novos, tendo o modelo sido definido no início dos anos 90. Algumas arquitecturas mais importantes no panorama actual dos sistemas de informação, que usam este modelo são: CORBA, DCOM e JavaRMI. No entanto, o modelo apenas tem sido adoptado parcialmente, no âmbito interno das organizações, devido à dificuldade de implementação dos conceitos de independência de localização, de flexibilidade dos dados e de interoperabilidade.

2.3 Exemplos de Arquitecturas Orientadas a Serviços

A computação distribuída tem vindo a assumir ao longo dos anos um papel de destaque no mundo dos sistemas de informação, devido à necessidade de integração das aplicações que se encontravam em ambientes heterogéneos e a necessidade de comunicação entre elas em tempo real.

Após algumas tentativas para arranjar um modelo para suprir estas necessidades, em 1988, o consórcio OSF, *Open Software Foundation*, lançou o DCE, *Distributed Computer Enviroment*, preconizando o primeiro passo para a computação distribuída. Todavia o DCE não usava o modelo de orientação a objectos.

Surgiu então o CORBA, *Common Object Request Broker Architecture*, que define uma arquitectura de objectos para a computação distribuída. Seguidamente surge o DCOM, *Distributed Component Object Model*, promovido pela Microsoft. Com a popularização da linguagem Java, surge o JavaRMI, *Java Remote Method Invocation*, lançado pela JavaSoft.

Desde o ano 2002, está a ser definida uma nova arquitectura, os Web Services, que surgem com grandes promessas de colmatar todos os problemas sentidos com as arquitecturas antecedentes.

Estas tecnologias baseiam-se numa arquitectura conhecida como arquitectura cliente-servidor de *stub/skeleton*. A principal finalidade é fornecer acesso programático a serviços remotos como se de uma entidade local se tratasse. Esta arquitectura torna a criação dos programas cliente relativamente simples porque os objectos e os seus métodos remotos são tratados exactamente como os locais. A própria arquitectura é responsável pelo transporte dos dados do cliente para o servidor e vice-versa. A figura 2.2 ilustra a arquitectura de *stub/skeleton*.

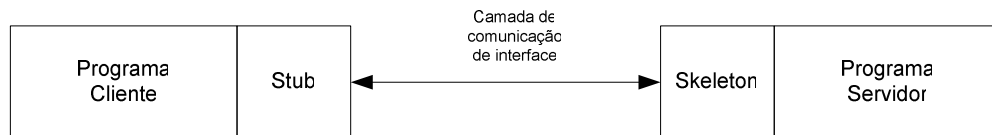


Figura 2.2 – Arquitectura cliente-servidor de *stub/skeleton*

O *stub* é um bloco de código localizado no computador do cliente. Basicamente tem três funções: a primeira, é identificar quais os objectos e métodos disponíveis no servidor; a segunda, codificar as chamadas de métodos, por parte do cliente, num formato que o servidor entenda; e a terceira, decodificar os dados que são retornados do servidor para um formato que o cliente entenda.

O *skeleton* realiza as mesmas funções do *stub*, mas na parte do servidor. Além disso, expõe quais as funcionalidades existentes no servidor que o cliente pode invocar, recebe as requisições do cliente e devolve os dados para o cliente.

Nos próximos pontos, vamos abordar as principais características das tecnologias que se baseiam nestas arquitecturas, através de uma análise comparativa; no entanto, reservamos o capítulo três para abordar os conceitos da nova arquitectura emergente, os Web Services.

2.3.1 CORBA

A especificação CORBA foi definida em 1990 pelo OMG, *Object Management Group*, um grupo formado por mais de 800 empresas de tecnologias de informação. Este consórcio iniciou um projecto de normalização, que procurava criar um ambiente universal de desenvolvimento de aplicações cliente-servidor em ambientes heterogéneos, baseadas em objectos distribuídos.

A base desta arquitectura é um barramento de objectos, denominado ORB, *Object Request Broker*, ao qual os componentes de *software* se ligam para interagirem entre si, como demonstrado na figura 2.3. O ORB é responsável por fornecer uma estrutura de comunicações que permite aos objectos dialogarem, independentemente de aspectos específicos da plataforma e das técnicas utilizadas para a sua implementação, isto é, um cliente pode invocar de forma transparente um método de um objecto no servidor, o qual pode estar no mesmo computador ou algures na rede, pois o encaminhamento do pedido é da competência do ORB [Roque, 2003].

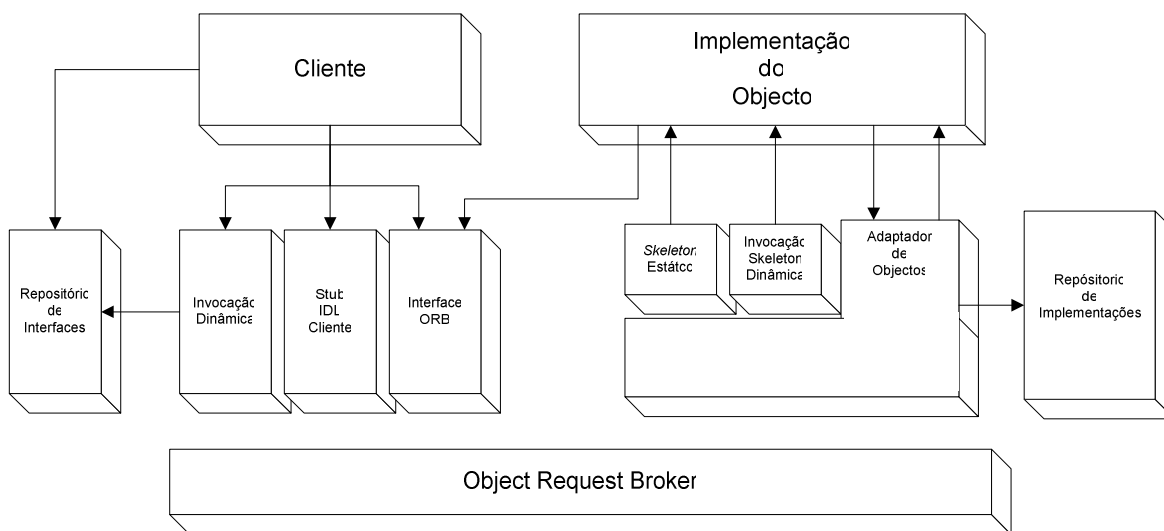


Figura 2.3 – Arquitetura do ORB do OMG

Principais características da arquitectura CORBA [Potts, 2003]:

- **Neutra relativamente à linguagem** – Foi desenvolvida para ser implementada com qualquer linguagem, recorrendo à linguagem neutra IDL, *Interface Definition Language*, para descrever a estrutura de todos os objectos;
- **Multiplicidade de fornecedores** – Actualmente existem muitos fornecedores de ORBs, o que permite que os clientes escolham os fornecedores que ofereçam as capacidades e custos que lhes sejam mais convenientes;
- **Implementação** – Invocação estática e dinâmica de objectos, independência da linguagem de alto nível, inventário das interfaces disponíveis através do Repositório de Interfaces, mensagens polimórficas e segurança.

A interoperabilidade é uma parte importante da especificação, não só na perspectiva dos programadores, mas também numa perspectiva de coexistência de múltiplos ORBs.

Na CORBA é conseguida com o recurso a um protocolo denominado GIOP, *General Inter-ORB Protocol*, que define as mensagens possíveis e a respectiva sintaxe, sendo por sua vez este protocolo suportado por mecanismos de transporte. Para o caso do TCP/IP o OMG definiu o IIOP, *Internet Inter-ORB Protocol*.

Em teoria a CORBA foi projectada para ser uma excelente solução para sistemas cliente-servidor heterogéneos; no entanto, na prática, tem vindo a evidenciar alguns problemas. As primeiras versões desta especificação deixaram muitas áreas abertas à interpretação dos fornecedores e como resultado, os ORBs de diferentes fornecedores não conseguiam comunicar uns com os outros, limitando a capacidade de combinação e interoperabilidade.

Para a especificação ser multi-linguagem, é exigido que os programadores entendam a linguagem IDL e especifiquem todos os objectos que estão envolvidos nos pedidos.

Outro aspecto que coloca algumas reticências, especialmente aos administradores de rede, é o facto da CORBA requerer o uso de portas de comunicação especiais, que muitas vezes representam um grande risco de segurança.

2.3.2 DCOM

O DCOM, *Distributed Component Object Model*, é o mecanismo da Microsoft para realizar chamadas de procedimentos remotamente. Historicamente, o DCOM teve início em 1990, quando a Microsoft introduziu a tecnologia OLE, *Object Linking and Embedding*, para suportar a integração de várias aplicações e diferentes tipos de dados multimédia numa mesma ferramenta de composição de documentos. Esta tecnologia tinha como suporte o DDE, *Dynamic Data Exchange*, mas apresentava bastantes limitações.

Na versão 2 do OLE de 1993, a maioria das deficiências da versão anterior foram resolvidas recorrendo a uma nova tecnologia de encapsulamento de objectos designada COM, *Component Object Model*. O COM passa então a ser publicitado pela Microsoft como um modelo de programação orientado por objectos e desenhado para facilitar a interoperabilidade de *software*. Na figura 2.4 podemos analisar a arquitectura DCOM [Roque, 2003].

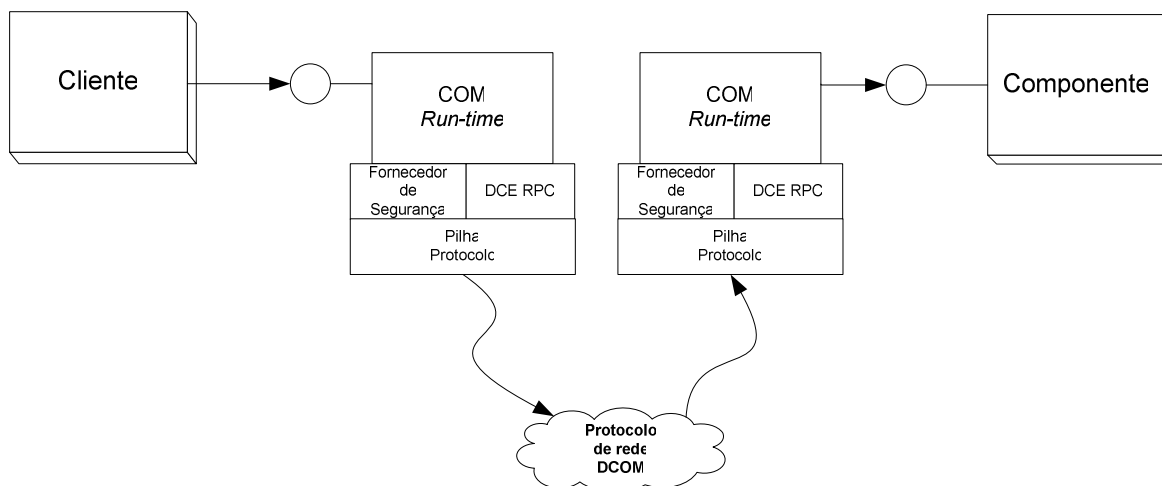


Figura 2.4 – Arquitectura DCOM

Tal como a maioria dos sistemas baseados em objectos, também o DCOM recorre a uma linguagem de definição das interfaces dos seus objectos. No DCOM não houve a preocupação de definir ligações com linguagens de alto nível, facto que se revelou como um factor limitativo desta tecnologia. A solução encontrada pela Microsoft para resolver o problema da interoperabilidade foi o recurso a uma norma binária. Desta forma, qualquer linguagem que entenda esta norma, pode criar e utilizar objectos DCOM. O número de linguagens e ferramentas que suportam o DCOM tem vindo a aumentar, e a Microsoft disponibiliza ligações para todos os seus ambientes de desenvolvimento.

Embora o DCOM possa ser implementado por diversas linguagens de programação, só poderá ser utilizado em plataformas Microsoft, o que significa que as duas extremidades envolvidas na transacção, o cliente e o servidor, têm de ser sistemas baseados em tecnologias da Microsoft.

2.3.3 JavaRMI

A linguagem Java é um caso de sucesso e os programadores desta linguagem têm a possibilidade de desenvolver aplicações distribuídas sem terem que se preocupar com as plataformas que as vão suportar, sendo apenas necessário que as mesmas suportem JVM, *Java Virtual Machine*. A especificação JavaRMI é o mecanismo da Sun para chamadas de procedimentos remotos.

A JavaRMI tira partido do ambiente de operação Java na construção de aplicações cliente-servidor, encontrando-se integrada no JDK, *Java Development Kit*.

Ao contrário das duas arquitecturas apresentadas anteriormente, a arquitectura JavaRMI é dependente da linguagem que lhe dá origem, o Java. Portanto não há necessidade de IDL.

A JavaRMI recorre na sua implementação a um modelo de três níveis, como pode ser visto na figura 2.5, em que cada um dos níveis é independente do seguinte, podendo, desta forma, cada nível ser substituído por implementações alternativas, sem que tal facto afecte os outros. Os três níveis são [Roque, 2003]:

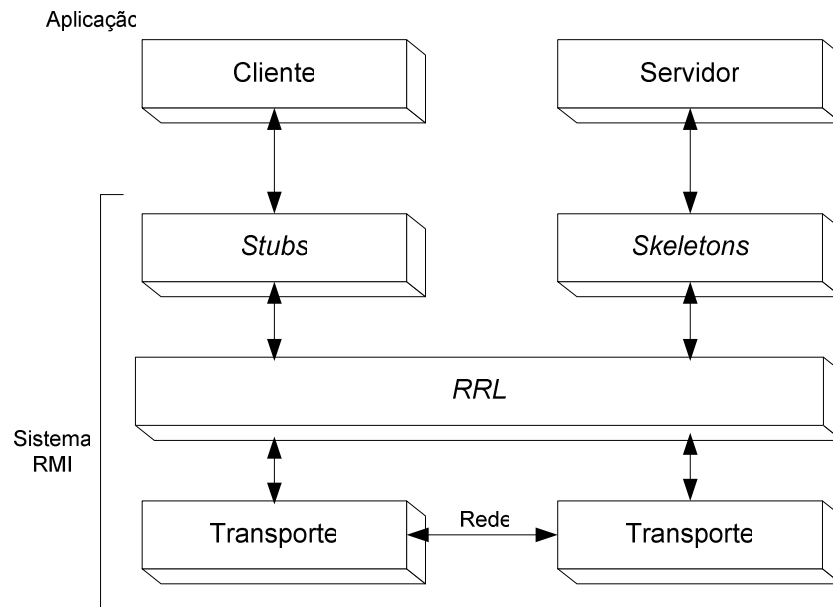


Figura 2.5 – Arquitectura JavaRMI

1º nível – Stub/Skeleton – responsável pela gestão das interfaces dos objectos remotos do cliente e do servidor;

2º nível – RRL, *retransmission request list* – responsável pela gestão da actividade dos objectos remotos, tendo também como função a gestão das comunicações entre os clientes e servidores com as JVMs para objectos remotos;

3º nível – Transporte – responsável pela comunicação entre clientes e servidores.

Um factor importante e limitativo desta tecnologia situa-se ao nível do protocolo de transporte utilizado, o RMP, *Remote Method Protocol*. Este protocolo, propriedade da Sun, não foi desenhado com o objectivo de interoperar com outros ORBs e linguagens. Este facto limita substancialmente a JavaRMI relativamente ao seu uso como tecnologia de suporte dos *backbones* de Intranets e da Internet.

Para superar esta limitação as empresas Javasoft/Sun e o OMG estão a trabalhar em conjunto na convergência dos dois modelos de objectos. Esta convergência passa, por exemplo, por a JavaRMI utilizar o protocolo IIOP do CORBA em substituição do RMP.

2.3.4 Web Services

Esta tecnologia, suporta o trabalho desta tese, aparece como uma verdadeira revolução na realização da integração de sistemas heterogêneos de uma forma simples, usando normas abertas e com grande sucesso na Internet, podendo substituir as tradicionais estratégias EAI, *Enterprise Application Integration*, que, resultam de uma tentativa que as empresas fizeram para interligar as aplicações de negócios internas, e para partilha dos dados. Recentemente, esta definição expandiu-se, significando agora também a união dos dados e a ligação com parceiros externos.

Basicamente, um Web Service funciona como uma página *Web*, com a diferença de que, ao invés de HTML, se utiliza XML e não tem interface gráfica. O XML é linguagem natural dos Web Services, ou seja, os fornecedores de serviços descrevem os seus serviços, ou seja as suas funções, em XML, os clientes requisitam os serviços usando XML, os serviços são descobertos através de XML, e quaisquer dados retornados para o cliente são em formato XML. Desta forma, os dados podem ser descritos, e o pacote da mensagem SOAP, *Simple Object Access Protocol*, pode ser manipulada com grande facilidade, tanto por quem envia, quanto por quem recebe. Na figura 2.6 podemos ver o funcionamento lógico de um Web Service.

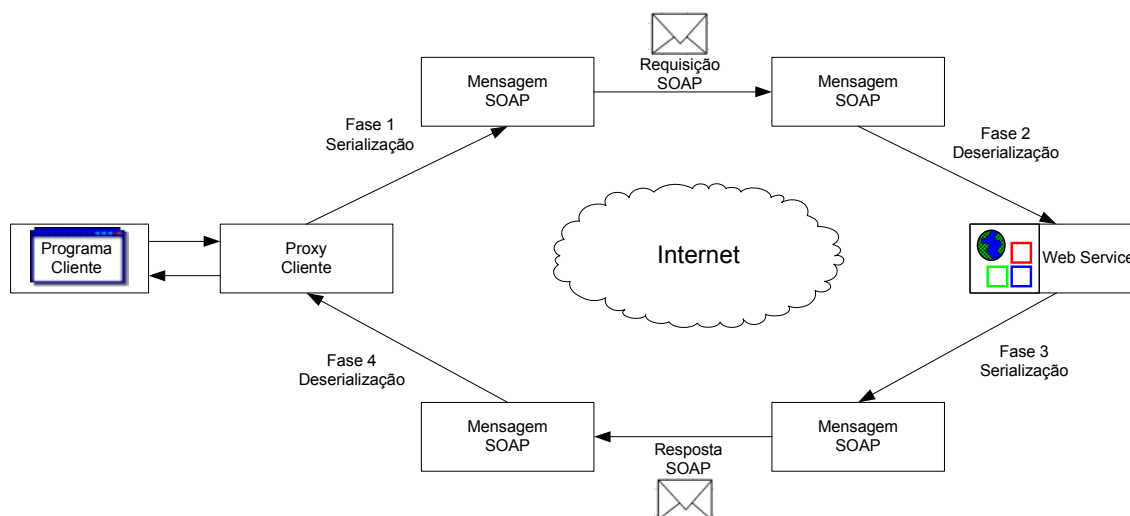


Figura 2.6 – Funcionamento lógico de um Web Services

Em termos práticos, a implementação dos conceitos da Arquitectura Orientada a Serviços tem sido realizada através da arquitectura dos Web Services. Isto acontece pelo facto de ser uma solução fortemente baseada na linguagem XML [W3C, 2005b], que é de ampla aceitação, e tem-se consolidando como a alternativa mais adequada na integração de sistemas heterogéneos.

A tecnologia que os Web Services oferecem é a primeira com potencial para dinamizar a implementação de aplicações distribuídas segundo o modelo da Arquitectura Orientada a Serviços em ambientes de múltiplas organizações. Assim, o modelo da Arquitectura Orientada a Serviços ganha uma nova vida, que tudo indica ser próspera, isto porque, por um lado, as principais normas abertas estão em plena fase final de definição, e, por outro lado, a indústria está entusiasmada e tem trabalhado em prol do mesmo objectivo.

2.4 CORBA, DCOM e JavaRMI versus Web Services

Nos pontos anteriores analisamos as tecnologias CORBA, DCOM e JavaRMI, que são os exemplos mais maduros e amplamente usados na construção de aplicações computacionais distribuídas.

Contudo, mais uma vez, no mundo da tecnologia informática, a história repete-se: toda a comunidade técnica se mobiliza para entender os detalhes e as implicações do aparecimento de uma nova tecnologia, tanto mais, quando a tecnologia representa uma grande mudança nos paradigmas de desenvolvimento de aplicações distribuídas, rompendo com os conceitos do passado. Desta feita, os Web Services surgem com grandes promessas e apresentam argumentos suficientemente capazes para se imporem e resolverem todos os problemas de que as outras tecnologias padecem.

Nos pontos seguintes, cada uma destas tecnologias tradicionais, CORBA, DCOM e JavaRMI, serão confrontadas directamente com os Web Services, destacando em que contextos essas tecnologias tradicionais levam vantagem sobre os Web Services, assim como em que âmbitos perdem para os Web Services.

CORBA versus Web Services

No que concerne ao suporte a múltiplas linguagens e múltiplas plataformas, as soluções CORBA são tão competente quanto as soluções Web Services.

No entanto, a CORBA usa a IDL como linguagem para traduzir os dados, os Web Services usam em toda a sua arquitectura a linguagem XML, a qual por se tratar de texto claro, é facilmente interpretada por humanos e máquinas.

A CORBA pode manipular cargas de transacções mais altas, uma vez que mantém uma conexão persistente entre um cliente e um servidor.

Em termos de maturidade, a CORBA leva vantagem devido à sua história, e muitos dos problemas iniciais de falta de interoperabilidade já foram resolvidos.

DCOM versus Web Services

Apesar do DCOM ser suportado por diversas linguagens, é menos flexível do que os Web Services, devido à forte ligação com a plataforma Microsoft. Os Web Services podem ser implementados com ferramentas de fornecedores diferentes em várias plataformas.

Comparando com os Web Services, as soluções DCOM são quase tão capazes para desenvolvimento inter-linguagem, mas normalmente são limitadas a uma estrutura totalmente Microsoft.

JavaRMI versus Web Services

As soluções JavaRMI, na especificação original, vêm limitada a sua utilização a um ambiente onde cliente e servidor usem a plataforma Java.

Uma vez que usam portos de rede específicos, a sua implementação torna-se mais complexa do que uma solução com Web Services.

Podem manipular maiores cargas de transacções, porque mantêm uma conexão persistente entre um cliente e um servidor.

Comparando com os Web Services, as soluções JavaRMI são a melhor opção, se as duas extremidades envolvidas na comunicação se basearem na plataforma Java. No entanto, torna-se praticamente inútil em plataformas não Java.

2.5 Necessidade de uma nova arquitectura

Uma vez que as entidades promotoras da CORBA, DCOM e JavaRMI já tiveram alguns anos para o seu aperfeiçoamento, será que uma nova tecnologia, que, na essência, é uma concorrente, terá razão de existir e mercado no mundo das tecnologias de informação? A resposta afirmativa a esta questão é a prova da insatisfação com as tecnologias anteriores e alguns pontos essenciais podem determinar a utilização de Web Services em seu detrimento. Algumas características negativas das arquitecturas CORBA, DCOM e JavaRMI são: [Potts, 2002]

- São orientadas para chamadas síncronas a métodos de objectos distribuídos;
- Codificam os dados em formato binário;
- A validação dos dados é feita através de programação;
- São concorrentes, mas não são interoperáveis;
- A feroz competição tornou impossível o consenso;
- Os interesses dos fornecedores destas tecnologias são, naturalmente, antagónicos, que faz com que a incompatibilidade entre as arquitecturas seja propositada.

Obviamente, uma nova solução, para ter sucesso, teria que evitar os problemas de que estas tecnologias padecem. Seguidamente, são apontadas algumas características positivas dos Web Services: [Potts, 2002]

- Estão a ser definidos, desde o início, tendo por objectivo principal a integração de aplicações, o que permite que suportem transacções síncronas e assíncronas;
- A codificação dos dados é feita usando tipos de dados do esquema XML. Cada lado da transacção decide como representar esses dados no código da aplicação. Os documentos XML são legíveis quer para humanos, quer para máquinas;
- Teoricamente, os Web Services são interoperáveis e a prática tem comprovado essa capacidade. A arquitectura está projectada para que a interoperabilidade aumente ao longo do tempo;
- Esta nova solução fica definitivamente marcada por uma surpreendente cooperação entre os fornecedores de tecnologias. Como os Web Services são definidos abstractamente, esse facto permite que sejam implementados praticamente em qualquer ferramenta de *software*;

- A definição da especificação não está a cargo de apenas uma empresa, com os seus interesses naturais de mercado, mas de vários consórcios, empenhados em encontrar as melhores soluções e evitando os erros do passado.

A tabela 2.1 apresenta de forma resumida, para cada uma das tecnologias abordadas nos pontos anteriores, como implementa as principais características que fazem uma tecnologia ser considerada mais apta para soluções distribuídas.

Tabela 2.1 – Comparação entre as várias arquitecturas

Característica	Tecnologia			
	CORBA	DCOM	JavaRMI	Web Services
Mecanismo RPC	IIOP (<i>Internet-ORB Protocol</i>)	DCE-RPC (<i>Distributed Computing Environment / Remote Procedure Call</i>)	RMP (<i>Remote Method Protocol</i>) / (<i>IIOP Internet-ORB Protocol</i>)	HTTP (<i>HyperText Transfer Protocol</i>) SOAP (<i>Simple Object Access Protocol</i>)
Codificação	CDR (<i>Common Data Representation</i>)	NDR (<i>Network Data Representation</i>)	Java	XML (<i>Extensible Markup Language</i>)
Descrição da Interface	IDL (<i>Interface Definition Language</i>)	IDL (<i>Interface Definition Language</i>)	Java-Java - não tem Java-CORBA - IDL	WSDL (<i>Web Service Description Language</i>)
Divulgação	Serviço de nomes e registo	Registo	Registo (<i>RMIregistry</i>)	UDDI (<i>Universal Discovery Description and Integration</i>)
Tolerante a Firewall	Não	Não	Não	Sim
Complexidade dos Protocolos	Alta	Alta	Alta	Baixa
Multi Plataforma	Parcial	Não	Não	Sim

Após a leitura do presente capítulo, rapidamente nos apercebemos de que a arquitectura dos Web Services está a ser construída sobre alicerces tecnológicos sólidos, o que significa que estão a ser evitados todos os erros do passado. Toda a indústria de *software* está a encarar com grande entusiasmo este novo conceito dos Web Services, devido à grande promessa de interoperabilidade.

No próximo capítulo, vamos analisar em detalhe todas as especificações, diferentes mas cooperantes, que, em conjunto, formam esta nova arquitectura a que chamamos Web Services.

Web Services

3.1 Introdução

Nos últimos anos, a Internet tem vindo a aumentar a sua importância e a sua aplicabilidade na computação distribuída. As crescentes potencialidades foram em grande parte impulsionadas com a criação da linguagem XML, que é cada vez mais usada para estruturar e representar todo o tipo de informação, afigurando uma nova abordagem de toda a arquitectura da Internet. Por outro lado, devido à sua simplicidade e omnipresença, a Internet representa a escolha lógica para a espinha dorsal de comunicação entre aplicações distribuídas.

No entanto, a infra-estrutura da Internet foi inicialmente projectada para consumo humano, isto é, a informação e interacção era normalmente requerida por utilizadores humanos, utilizando, tipicamente, *browsers*.

Algumas limitações resultam dessa característica. Consideremos o seguinte exemplo: uma empresa que diariamente tem que consultar o preço de um produto em vários fornecedores e, depois, comprar determinada quantidade do produto ao fornecedor que ofereça o melhor preço. Um funcionário da empresa pode realizar diariamente esta tarefa, acedendo aos sistemas *Web* de comércio electrónico dos fornecedores. No entanto, como se trata de uma tarefa diária, seria no mínimo interessante que a mesma fosse automatizada e se transformasse numa rotina do sistema de informação de compras da empresa.

Para tal, seria necessário que esse sistema de compras estivesse integrado com os sistemas de informação dos fornecedores. Entretanto, utilizando os protocolos e linguagens originais da *Web* esta integração é praticamente inexecutável. Esta dificuldade resulta do facto de as informações que saem de um sistema *Web* tradicional, serem documentos

textuais, os quais, são facilmente interpretados por utilizadores humanos, através de um *browser*, mas não por outras aplicações computacionais.

Actualmente, encontra-se em desenvolvimento uma extensão do modelo tradicional da *Web*, com o objectivo de disponibilizar dados estruturados que serão utilizados, ou seja, consumidos por programas escritos em qualquer linguagem de programação. Esta extensão, conhecida pelo nome de Web Services, apresenta a forma de dotar os servidores e clientes *Web* de programas com capacidade de processar mensagens XML.

Os programas implementados com o conceito de Web Services têm como entrada e saída de dados mensagens estruturadas em XML. Por exemplo, no caso do sistema de compras anteriormente descrito, cada fornecedor poderia disponibilizar num servidor *Web* um programa, isto é, um Web Service, capaz de:

- Processar um documento XML de entrada que descreve o produto, com o código e quantidade;
- Obter o preço do produto solicitado, no sistema de informação da empresa, ou invocar outro Web Service;
- Devolver como resposta, um outro documento XML, contendo o preço do produto.

O cenário descrito pode ser observado na figura 3.1, onde podemos analisar a interacção entre a aplicação cliente e os Web Services dos fornecedores do produto, para consulta do preço de um determinado produto, culminando com a encomenda do produto ao fornecedor que oferece melhor preço. Toda esta interacção processa-se de forma automática sem intervenção do utilizador.

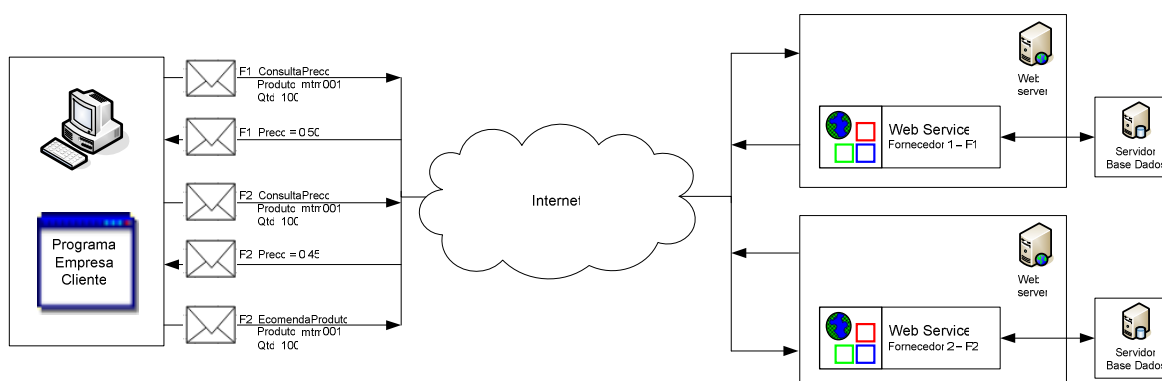


Figura 3.1 – Cenário de utilização de Web Services

3.1.1 Problema: partilha de dados

Actualmente, praticamente todas as organizações possuem sistemas de informação mais ou menos complexos. Da mesma forma que um novo funcionário tem necessariamente que se relacionar com os funcionários actuais, as aplicações adquiridas de novo deviam também ter a capacidade de se integrarem com as já existentes no sistema de informação.

Entretanto, as organizações não existem nem podem funcionar isoladas no mundo. Usando uma analogia parecida com a anterior, tal como os responsáveis de cada organização interagem com os vários parceiros de negócio para concretizar a missão da organização, as aplicações dessas organizações deviam também ter a potencialidade de interagirem entre elas.

No entanto, a história da computação mostra que grande parte do desenvolvimento de sistemas de informação tem ocorrido numa lógica de estes serem usados a nível do computador pessoal e em redes locais, portanto, dentro de uma área geográfica limitada ao perímetro da organização.

Ao longo da última década do século XX, várias arquitecturas foram propostas, mas nenhuma delas apresentou características que lhe permitisse vingar como uma solução globalmente aceite. O estudo dessas principais tecnologias foi assunto do capítulo dois.

Apesar da grande evolução dos sistemas de B2B, *business-to-business*, a componente de integração de sistemas, que de facto é o que está na génese do conceito de B2B, ainda se mantém como um dos maiores obstáculos a serem vencidos. Parte considerável do problema acontece por uma razão simples: o trabalho de integrar aplicações é feito ao contrário. Primeiro, desenvolvem-se os sistemas a fim de responder a necessidades imediatas do negócio, depois detecta-se que a organização não é uma ilha isolada e que precisa de comunicar com as outras, o problema surge mesmo entre departamentos internos.

Conclui-se então que o cenário ideal era um ambiente onde fosse possível a interacção automática entre aplicações, utilizando a infra-estrutura mundial que a *Web* disponibiliza. Um grupo de trabalho do W3C declara a este respeito:

“...A Web pode incrementar significativamente o seu poder e âmbito se for estendida para suportar a comunicação entre aplicações, de um programa para outro.” – W3C XML Protocol Working Group [W3C, 2005b].

Neste contexto surgiu uma nova tecnologia, que além de outros usos, também pode ser utilizada para integração de aplicações, quer internas ou externas. Essa tecnologia traduz-se no conceito de Web Services.

3.1.2 A nova Internet

Uma década após o aparecimento da Internet, após as transformações sofridas, que fizeram parte do seu processo de evolução, podemos resumir as suas principais potencialidades actualmente, nos seguintes pontos:

- Troca de informação;
- eCommerce B2C, *Business to Consumer*;
- Interacção B2B, não automatizadas.

Da análise dos pontos anteriores, podemos concluir que, actualmente, a maior parte das actividades efectuadas no ambiente da Internet são feitas numa lógica de interacção homem – aplicações.

Com a globalização que está a acontecer em todas as áreas, novas necessidades surgem, especialmente a nível de negócios que estão a ser criados de novo ou então a serem remodelados para utilizarem esta plataforma planetária que é a Internet. Portanto, é essencial aproveitar o clima actual favorável da Internet e dos seus protocolos, para a utilizar como condutora de novas funcionalidades fundamentais para as organizações, agregando assim, conceitos como A2A, *application-to-application*, ou seja interacção entre aplicações e EAI, *Enterprise Application Integration*, facilitando assim, a integração de sistemas de informação entre empresas.

O criador da *Web*, Tim Berners-Lee, como já referido no capítulo um, aborda o potencial da *Web* segundo duas perspectivas: a primeira, ligada à comunicação e interacção entre pessoas, o que acontece actualmente em grande escala; a segunda, como um espaço de comunicação entre aplicações, onde agentes de *software* podem actuar em nosso nome, dentro de limites estabelecidos, para facilitar o dia-a-dia e libertar-nos de tarefas fastidiosas e repetitivas.

Actualmente, grande parte dos esforços dos grupos de trabalho do W3C estão centrados no intitulado terceiro grande salto da *Web*, que acontecerá com a introdução do conceito de *Web* semântica que consiste, basicamente, em agregar dados sobre os dados,

para que os agentes de *software* possam interpretar os dados, melhorando desta forma as interacções com a Internet, quer por parte de pessoas, quer por parte das aplicações.

Complementarmente, os Web Services estão rapidamente a tornar-se numa tecnologia significativa na evolução da *Web* e da computação distribuída. Esta tecnologia alcançou a independência dos dados introduzida pelo XML, permitindo por exemplo, resolver problemas de integração de aplicações entre empresas. As interfaces dos Web Services funcionam como invólucros que fazem a associação dos dados independentes para qualquer tipo de aplicação, *middleware* ou sistema de base de dados. Finalmente, tudo parece conjugar-se para a concretização da segunda potencialidade anunciada por Tim Berners-Lee, permitindo assim a completa cooperação entre pessoas e agentes computacionais.

Para a concretização deste dinamismo entre agentes computacionais, a arquitectura da *Web* terá que agregar um novo paradigma de orientação a dados. Hoje, a maioria das interacções na *Web* surge de uma diligência por parte do utilizador, usando tipicamente, um navegador da *Web*, comumente conhecido por *browser*. Os sítios *Web* estão programados para responder a requisições de um utilizador, a resposta normalmente é texto no formato HTML que orienta a apresentação do conteúdo no *browser* do cliente.

Com a introdução do paradigma da *Web* orientada a dados, a informação trocada entre cliente e servidor será constituída por dados, que podem ser interpretados e processados, utilizando para isso uma linguagem independente de qualquer plataforma, o XML.

Na tabela 3.1 comparamos as aplicações baseadas no conceito de Web Services com as aplicações *Web* tradicionais.

Tabela 3.1– Web tradicional versus Web Services

	Web tradicional	Web Services
Informações	Não estruturadas, para consumo humano	Estruturadas, para consumo de aplicações
Linguagem	HTML	XML, WSDL
Protocolos	HTTP	HTTP, SOAP, UDDI
Clientes	Browsers	Aplicações escritas em qualquer linguagem
Tipo de interacção	Homem – Máquina	Aplicação – Aplicação

3.2 Definição de Web Service

Traduzindo Web Services, obtemos Serviços Web em português. Ao longo deste trabalho, no entanto, optamos manter o termo em inglês para evitar a ambiguidade que o termo pode trazer em português. Esta ambiguidade reside no facto de se poder imaginar, qualquer serviço realizado através da *Web*, como por exemplo, o correio electrónico, como um Serviço Web.

Apresentar uma definição para o conceito de Web Services torna-se uma tarefa difícil, pois, por um lado, essa definição depende do perfil da pessoa ou organização que a vai apresentar, ou seja, se tem uma visão mais conceptual ou mais técnica e, por outro lado, como o conceito encerra um conjunto de outros paradigmas já existentes e de sucesso no contexto da Internet, a definição implica o relacionamento com estes. Por isso, surge a dúvida: será o conceito dos Web Services uma novidade, ou é o reinventar da roda, tentando dar uma nova vida a conceitos do passado.

Seguidamente apresentamos uma definição para o conceito de Web Services, produzida pelo W3C, que é uma das organizações mais importantes no contexto deste novo paradigma.

“Um Web Service é um sistema de software projectado para suportar interacção aplicação-a-aplicação sobre uma rede. Possui uma interface descrita num formato processável por máquinas, especificamente WSDL. Outros sistemas interagem com o Web Service de uma maneira aconselhada por esta descrição usando mensagens SOAP, tipicamente transportadas usando HTTP com uma serialização XML em conjunção com outras normas relacionadas com a Web” – W3C [W3C, 2005e].

Conceptualmente, os Web Services representam uma nova forma de programar aplicações distribuídas, aproveitando a infra-estrutura e a flexibilidade que a Internet oferece. Representa também, a forma de organizar uma pilha de protocolos normalizados da Internet, como HTTP, XML e SOAP, que de forma encadeada permitem activar a comunicação entre quaisquer aplicações no mundo, derrubando as barreiras tradicionais, de localização geográfica, sistema operativo, plataforma de hardware, linguagem de programação e protocolos.

Em parte, esta independência é conquistada pela utilização de uma linguagem universal, XML, que constitui a linguagem natural dos Web Services para todas as trocas

de dados entre os parceiros na comunicação e por isso, muitas das vezes, os Web Services, também são designados por XML Web Services.

Tecnicamente, podemos definir um Web Service como uma aplicação, sem interface gráfica, que expõe via *Web*, um conjunto de funções ou operações, que podem ser publicadas, num formato específico, para que outras aplicações ou Web Services as possam descobrir e invocar.

O acesso a um Web Service é feito programaticamente, normalmente, através de um URI, ou seja, um programa ou outro Web Service, usando uma mensagem, criada no formato XML, envia uma requisição a um Web Service pela rede e recebe normalmente uma resposta, também na forma de um documento XML. As normas relacionadas com os Web Services definem o formato da mensagem, especificam a interface para onde a mensagem é enviada, descrevem as convenções para o mapeamento do conteúdo das mensagens de entrada e de saída das aplicações que implementam o serviço, ou seja, as funções ou operações, e definem mecanismos para publicar e descobrir dinamicamente interfaces de Web Services.

Na figura 3.2 podemos analisar a interacção entre duas entidades: a aplicação, no lado do cliente, e o Web Service, no lado do servidor.

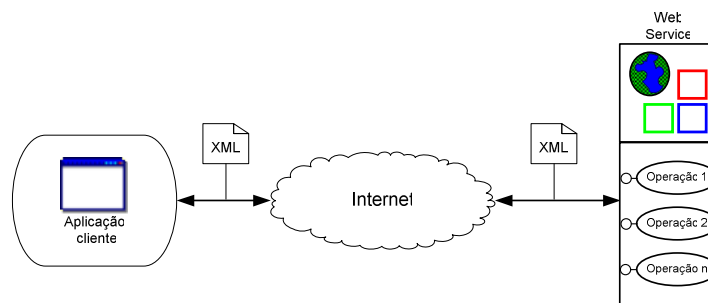


Figura 3.2 – Conceito de Web Service

Do lado do servidor, o Web Service, expõe um conjunto de operações, que por sua vez podem implementar qualquer função tradicional que permita aceder a qualquer sistema de informação da organização.

Do lado do cliente, a aplicação usa um mecanismo conhecido como *proxy* que referencia o Web Service, por meio do qual é possível chamar qualquer operação do Web Service, como se de um objecto local se tratasse.

A interacção entre a aplicação cliente e o Web Service é efectuada através de mensagens normalizadas num formato XML. Portanto, o cliente coloca numa mensagem a indicação do método que pretende que seja executado no servidor e eventuais parâmetros.

O servidor depois de receber a mensagem executa a operação referenciada e devolve a resposta da execução da operação ao cliente.

Na figura 3.3 está representado o processo de desenvolvimento de aplicações utilizando a invocação de procedimentos remotos através de Web Services. Como se pode observar na figura o cliente usa programaticamente os métodos que o servidor disponibiliza, como que de métodos locais se tratassem.

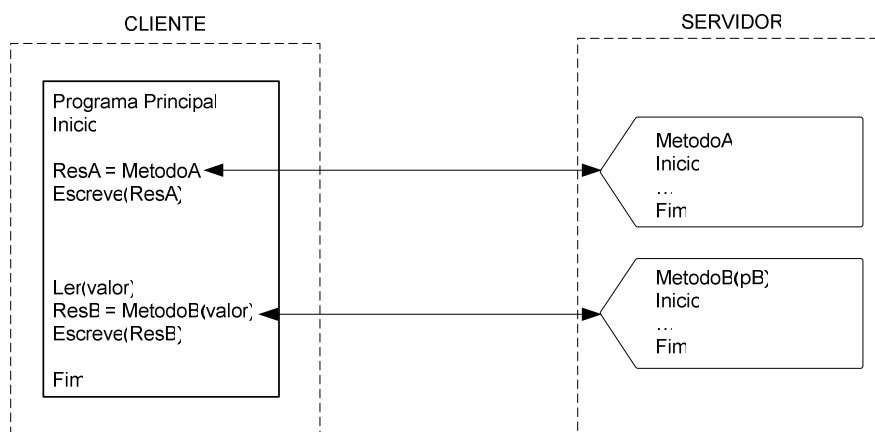
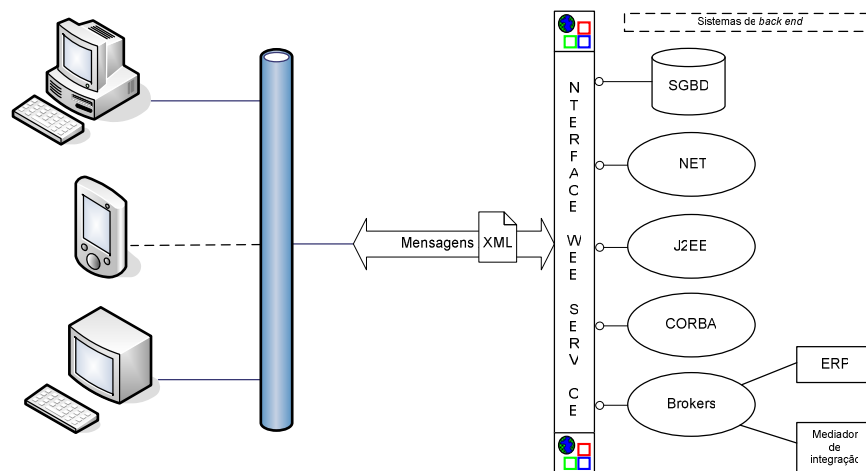


Figura 3.3 – Invocação de procedimentos remotos

A programação da aplicação cliente, é, normalmente, elaborada recorrendo a um conceito conhecido como *proxy*, que tem basicamente dois objectivos principais. O primeiro, é permitir o estabelecimento da ligação entre a aplicação cliente e o Web Service, e o segundo, é encapsular toda a complexidade inerente às mensagens de comunicação entre ambos, para troca dos dados e dos protocolos de transporte a utilizar. Basicamente, o que a aplicação cliente precisa fazer é ligar-se ao *proxy* e este estabelece o contacto com o Web Service propriamente dito.

A tecnologia oferecida pelos Web Services pode ser usada em muitos contextos, por exemplo, criar empresas totalmente virtuais, dando origem a uma nova categoria de aplicações baseadas na *Web* ou para resolver problemas de integração de empresas B2B, conectando aplicações de diversas organizações que participam na mesma cadeia de negócio. Também podem resolver problemas de integração de aplicações empresariais, substituindo as tradicionais técnicas de EAI, conectando aplicações de uma organização com outras aplicações, internas ou externas.

Em todos estes casos, os Web Services funcionam como um adaptador normalizado, ou seja, uma nova camada de *software*, para unir todos os artefactos das aplicações.

Figura 3.4 – Interface dos Web Services com sistemas de *back-end*

A figura 3.4 ilustra como os Web Services representam uma forma de criar uma interface normalizada entre sistemas de *back-end*, como sistemas de gestão de base de dados, .NET, J2EE, CORBA, adaptadores de pacotes ERP, *Enterprise Resource Planning*, e intermediários de integração e outros [Newcomer, 2002].

O conceito de Web Services tem a capacidade de combinar as características de programação de aplicações com as características de abstracção que a Internet possui. As tecnologias usadas actualmente na Internet alcançaram sucesso em parte porque foram definidas com um tal nível de abstracção que permitem a compatibilidade entre quaisquer sistema operativo, plataforma de hardware ou plataforma de *software*. Os Web Services ao usarem a infra-estrutura da Internet exploram este nível de abstracção e associam significado aos dados que se propagam através dela. Os Web Services não definem apenas os dados, definem também como processá-los e associa-los às aplicações respectivas, tanto nas mensagens de entrada como nas de saída.

3.3 Vantagens e desvantagens dos Web Services

O conceito dos Web Services surge com grandes promessas de derrubar todas as barreiras de interoperabilidade e constituem um grande avanço nos paradigmas de programação de aplicações distribuídas; por isso, o conceito tem sido encarado com grande entusiasmo por toda a comunidade de computação, quer por parte dos principais fornecedores de tecnologia quer por parte dos programadores.

Nos pontos seguintes apontamos as principais características que resultam em vantagens, agrupadas em características operacionais e características técnicas. [Potts, 2002]

Características operacionais:

- Menor custo operacional – as empresas estão constantemente a procurar formas de diminuir os custos associados ao negocio. Normalmente, essa procura, é focalizada nos sistemas internos e nos sistemas de relacionamentos com os parceiros comerciais. Os Web Services representam algumas formas para a concretização desses factores, por exemplo, ao utilizarem a Internet, que é a forma mais barata de interligar computadores, podendo assim, definir troca electrónica de dados e relatórios com clientes e fornecedores, eliminando assim, os sistemas baseados em papel;
- Menor custo de desenvolvimento – o facto dos Web Services utilizarem normas abertas permite-lhes serem interoperáveis independentemente da plataforma e linguagem, afigurando assim um grande avanço na reutilização de código;
- Integração com parceiros de negócio – a melhor forma de uma empresa se valorizar junto dos seus clientes é permitir que estes acedam aos seus sistemas de informação. Através de um Web Service possibilitar por exemplo que uma aplicação de um cliente possa alterar os seus dados, ou então, que consulte o preço de um determinado produto e, efectue encomendas directamente no sistema da empresa. Este pseudo controlo do sistema de informação, por terceiros, aumenta a precisão dos dados e dispensa recursos internos.

Características técnicas:

- Os Web Services podem ser criados e utilizados em qualquer plataforma e linguagem de programação, desde que, sejam capazes de processar documentos XML;
- Permitem comunicação síncrona e assíncrona;
- As mensagens podem ser enviadas utilizando o protocolo HTTP, permitindo assim a passagem das mensagens pelos sistemas de *firewall*, e não sendo portanto necessário definir portas especiais;
- Arquitectura fracamente acoplada, ou seja é independente de protocolos e códigos específicos para controle das transferências da informação;

- Reutilização dos serviços, porque os serviços existentes podem ser utilizados para compor um novo serviço;
- Modelo SOA, *Service-Oriented Architecture*, os Web Services são elaborados segundo o modelo de Arquitectura Orientada a Serviços no qual as aplicações desenvolvidas podem ser publicadas, descobertas e acedidas como um qualquer serviço disponível na *Web*;
- A definição de cada protocolo que compõe a arquitectura dos Web Services não depende apenas de uma empresa, mas sim de consórcios formados pelas principais empresas de tecnologia.

No entanto, nenhuma tecnologia é perfeita. Apesar dos Web Services prometerem resolver os problemas das tecnologias anteriores, eles próprios, podem padecer de alguns problemas. Alguns desses problemas são intrínsecos às fundações tecnológicas sobre as quais os Web Services assentam e outros resultam das suas próprias especificações.

Seguidamente, descrevemos alguns desses possíveis problemas:

- Disponibilidade – uma vez que os Web Services usam a infra-estrutura da *Web*, portanto, tal como um sitio da *Web*, isto é um endereço URL, podem não estar disponíveis a todo o momento.
- Garantia de execução – os Web Services por norma, usam o protocolo HTTP para transportar as mensagens, no entanto, o HTTP não garante a entrega e a resposta das mensagens. Por isso, os Web Services permitem usar outros protocolos de transporte, que resolvem esse problema.
- Desempenho – no contexto da Internet a única garantia que temos em relação ao desempenho é que este varia incontrolavelmente de um momento para o outro. O HTTP foi projectado para permitir que um servidor *Web* manipule muitas requisições, não mantendo, por isso, uma ligação persistente entre cliente e servidor, o que implica, inevitavelmente, desperdício de tempo. Por outro lado, não podemos ignorar também o tempo gasto no processamento dos documentos XML, pois, toda a comunicação é efectuada através de mensagens XML, num formato especial. Torna-se mais crítico, por exemplo, quando queremos enviar dados em formato binário, que terão de ser codificados para um formato que possa ser representado em XML.
- Segurança – é sem duvida a questão mais importante a ter em mente num ambiente distribuído, e susceptível a ataques, como é o caso da Internet. É nesta

área que se espera a finalização de várias especificações em curso que tratam de todos os tópicos relacionados com a segurança, como a validação, autenticação e não rejeição.

- Novidade da tecnologia – como os Web Services são uma tecnologia relativamente jovem, as normas e especificações evoluem rapidamente. As aplicações elaboradas com uma versão de uma especificação podem ficar rapidamente desactualizadas. Apesar dos Web Services basearem-se em normas abertas, cada fornecedor tem uma forma de as implementar, o que pode exigir uma grande actualização do sistema se mudarmos o conjunto de ferramentas para a construção de Web Services.

3.4 Arquitectura dos Web Services

O objectivo principal dos Web Services é activar a interacção automática entre aplicações. Assim, os dados, num formato interpretável por qualquer sistema computacional, ou até mesmo por humanos, podem ser transmitidos através da Internet por uma aplicação construída com a tecnologia de Web Service e recebidos por outra aplicação. Estes dados, uma vez no destino, depois de interpretados e processados podem ser utilizados da forma desejada, ou seja, programada na aplicação que os recebe. Por exemplo, podem ser apresentados numa página da Internet, numa aplicação *desktop* ou ainda num dispositivo móvel. Na figura 3.5 podemos analisar o ciclo completo de acções que ocorrem quando uma aplicação cliente pretende invocar uma função remota disponibilizada por um Web Service.

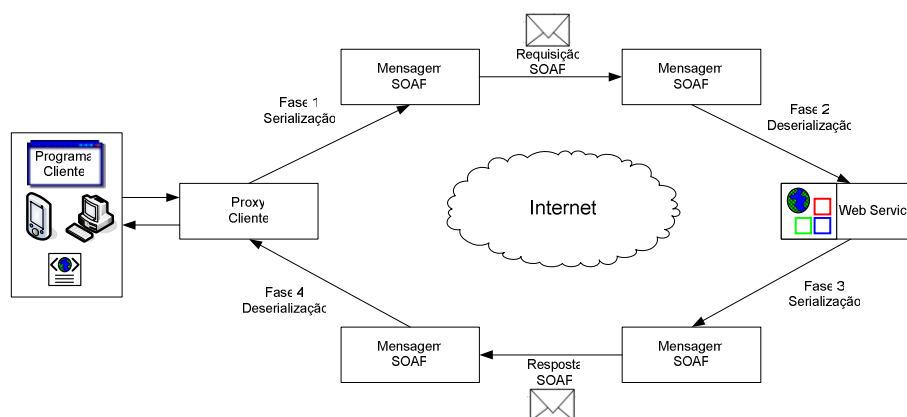


Figura 3.5 – Ciclo da interacção da aplicação cliente com o Web Service

A aplicação cliente invoca a função remota recorrendo ao *proxy*, este estabelece a ligação como o Web Service e empacota num envelope SOAP a mensagem com a informação necessária, que inclui o nome da função a executar remotamente e eventuais parâmetros. O Web Service, depois de receber a mensagem, executa a função requerida e, no caso de haver lugar a resposta, esta será enviada também num envelope SOAP.

Para activar ou obter um serviço é necessário comunicar com o Web Service. Os Web Services utilizam mensagens SOAP que são transmitidas através da *Web* usando um protocolo globalmente difundido, o HTTP. Por omissão, este protocolo usa a porta 80 da pilha protocolar TCP/IP. Esta porta, geralmente está activa para visualização de páginas *Web*, por isso, normalmente não está bloqueada por sistemas de segurança de uma *firewall*. Assim, para existir comunicação entre aplicação cliente e o Web Service, o que pode ocorrer em qualquer plataforma, bastando para isso que essa plataforma suporte SOAP e HTTP.

Portanto, na prática é através de mensagens SOAP que as aplicações cliente invocam as funcionalidades de um Web Service. Este processo designa-se por encapsulamento da mensagem e consiste basicamente em converter os objectos computacionais num documento XML, num processo conhecido por serialização. Gerada a mensagem SOAP, o protocolo HTTP é utilizado para fazer chegar a mensagem ao Web Service. Após a execução da função requisitada, o Web Service usando o mesmo processo envia a resposta à aplicação cliente.

O processo de encapsulamento do serviço é efectuado por um servidor SOAP que é composto por três módulos: gestor de serviços, lista de serviços e tradutor XML. No Web Service o servidor SOAP desempenha as seguintes funções:

- O gestor de serviços recebe requisições SOAP vindas de aplicações cliente. Verifica se o serviço solicitado existe no servidor SOAP. Caso afirmativo, entra em acção o tradutor XML;
- O tradutor XML converte a estrutura XML do pedido SOAP para um formato compatível com a chamada de método na linguagem de programação utilizada na construção do Web Service.

De forma análoga, após a obtenção da resposta do método do Web Service, essa resposta é convertida para XML, sendo de seguida empacotada numa mensagem SOAP. Após este processo, o tradutor XML devolve a resposta para o gestor de serviços que a faz chegar à aplicação cliente, através do protocolo HTTP, como pode ser visto na figura 3.6.

No essencial, o encapsulamento que ocorre quer na aplicação cliente quer no Web Service é similar.

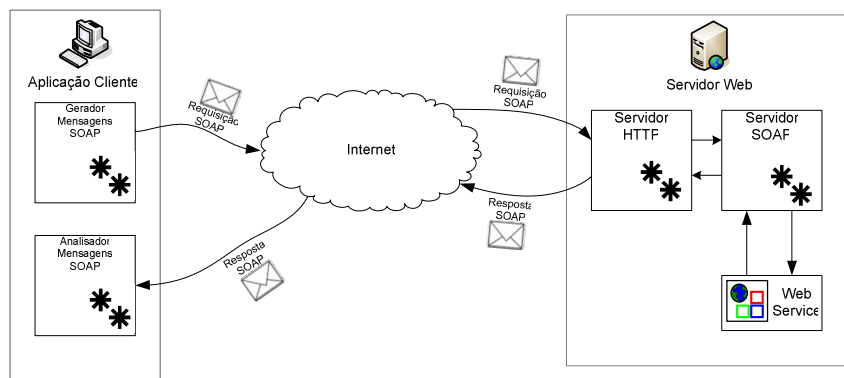


Figura 3.6 – Interação da aplicação cliente com o Web Service

3.4.1 Modelo de implementação

A Arquitectura Orientada a Serviços fornece o modelo teórico para a implementação de Web Services. Como analisado no capítulo dois, este modelo representa um novo modelo de desenvolvimento de aplicações, e constitui uma mudança para uma arquitectura de *software* relativamente distinta da arquitectura orientada ao objecto, hoje comum.

Segundo este modelo, as aplicações são elaboradas como entidades que expõem um conjunto de funções, os serviços. Este modelo define a interação entre três identidades, o fornecedor do serviço, o mediador do serviço e o requisitante do serviço, através de três operações: publicação, descoberta e ligação. A interação entre os elementos ocorre como se mostra na figura 3.7.

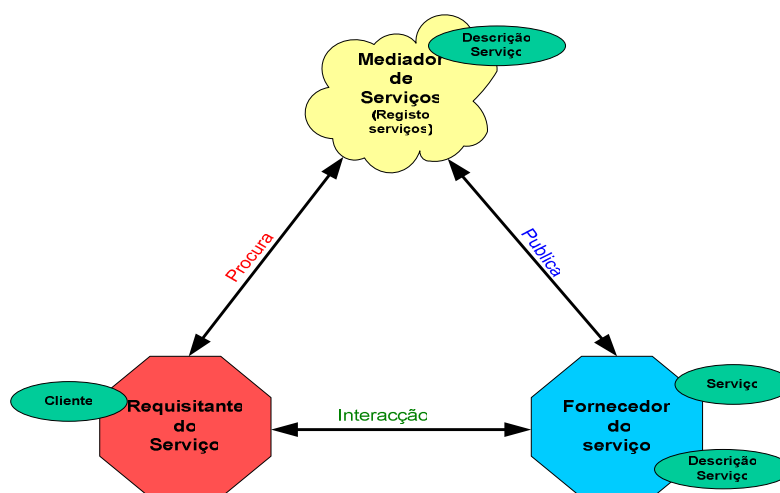


Figura 3.7 – Modelo SOA [W3C, 2005b]

A figura ilustra a arquitectura básica dos Web Services, na qual um requisitante do serviço, o cliente, e um fornecedor do serviço, o servidor, interagem baseados na descrição publicada pelo fornecedor e descoberta dinamicamente pelo requisitante através de um mediador de serviços.

O requisitante do serviço é uma aplicação que precisa usar um serviço de uma outra aplicação. No caso de não saber como localizar essa aplicação e nem quais as operações disponíveis, então recorre ao mediador de serviços usando a operação procura.

O mediador de serviço/registo de serviços consiste numa aplicação que também é um Web Service, que retorna informações em resposta a um critério de pesquisa que tenha sido submetido por um requisitante de serviço através da operação procura. Essas informações incluem dados necessários para contactar os Web Services, ou seja, com base na classificação de cada Web Service é retornada uma lista de Web Services que correspondem ao critério de pesquisa. Inclui também informações sobre como descobrir os detalhes de conexão para cada Web Service, permitindo assim, a criação automática de aplicações que podem usar os serviços disponibilizados por cada Web Service.

Os fornecedores de serviços publicam os detalhes de classificação, bem como os detalhes de conexão no mediador de serviços. O objectivo dessa publicação é o mesmo que tradicionalmente as empresas têm quando fazem publicidade nas páginas amarelas. O requisitante do serviço usa os detalhes de conexão para se ligar ao fornecedor do serviço. Uma vez ligados, estas duas entidades podem trocar mensagens, isto é, comunicam e interagem. A figura 3.8 ilustra o modelo descrito anteriormente.

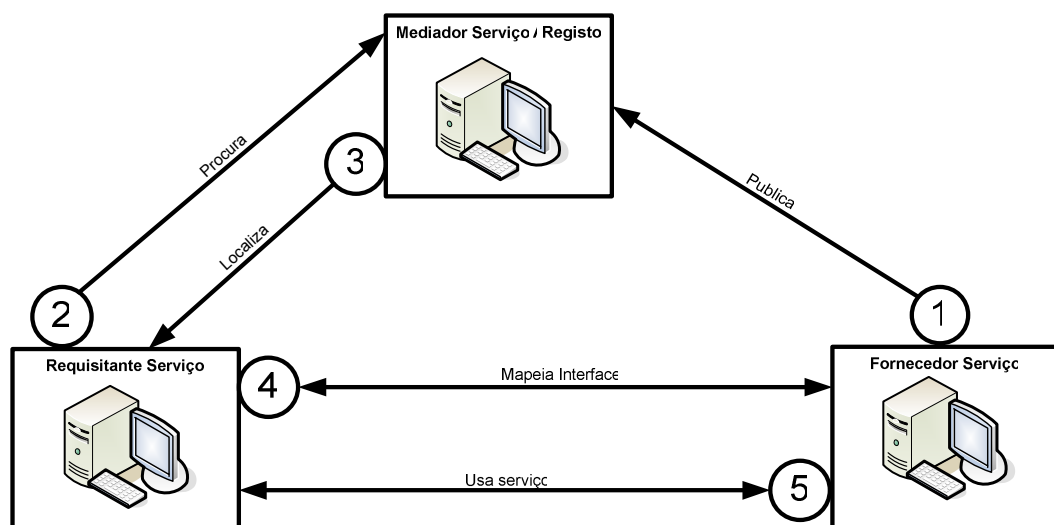


Figura 3.8 – Interação entre as entidades do modelo SOA

Na figura 3.8 podemos observar a sequência lógica das fases que constituem o modelo da Arquitectura Orientada a Serviços, que serve de suporte à construção de Web Services. Seguidamente, descrevemos as principais fases, desde a publicação até à utilização do Web Service por parte do requisitante do serviço. Essas fases são:

- **Fase 1** – o fornecedor do Web Service publica os serviços disponibilizados num registo, público ou privado, colocando o Web Service numa determinada categoria ou classificação, ficando assim o Web Service disponível para ser utilizado por potenciais clientes.
- **Fase 2** – o potencial cliente pode procurar os Web Services, num registo global centralizado, como que procurasse num catálogo de aplicações, o ou os que melhor dão resposta aos seus requisitos.
- **Fase 3** – o mediador de serviços devolve uma lista de Web Services que podem resolver as questões dos requisitos do cliente, juntamente com as informações necessárias para a construção de aplicações que usem esses Web Services.
- **Fase 4** – com as informações retornadas do mediador de serviços, o cliente pode vincular-se ao Web Service.
- **Fase 5** – uma vez vinculados, cliente e Web Services podem trocar mensagens, interagindo desta forma.

As fases descritas anteriormente podem ser observadas na figura 3.9 que a seguir se apresenta.

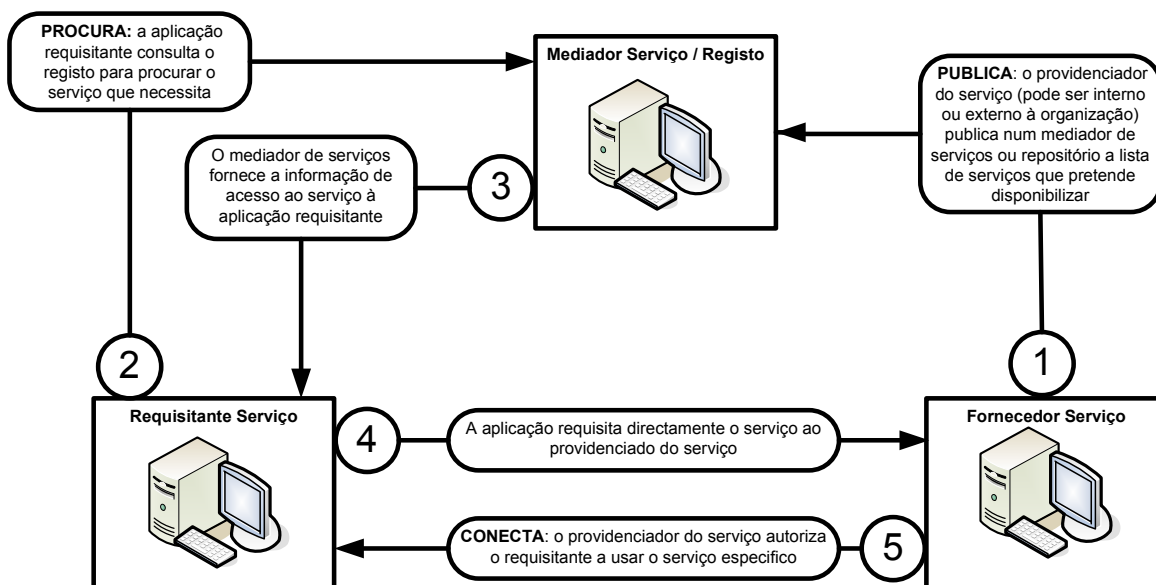


Figura 3.9 – Principais fases da interacção entre as entidades do modelo SOA

3.4.2 Base protocolar

Embora alguns autores reduzam o conceito de Web Services a uma fórmula bastante simples, ou seja, Web Services = XML + HTTP, evidenciando desta forma que a essência deste conceito consiste na utilização de uma linguagem normalizada, o XML, e de um protocolo para transporte, sendo o mais comum o HTTP.

No entanto, a arquitectura dos Web Services é composta por várias camadas de tecnologia interrelacionada, como pode ser observada na figura 3.10.

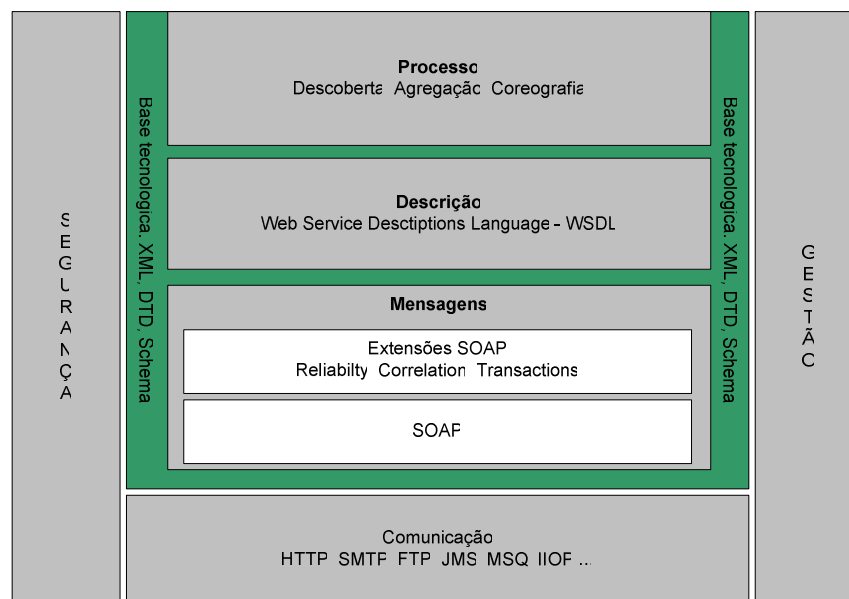


Figura 3.10 – Pilha de protocolos dos Web Services [W3C, 2005b]

Na pilha de protocolos apresentada na figura 3.10, podemos agrupar os protocolos em duas categorias principais: protocolos básicos e protocolos avançados.

Os protocolos básicos definem essencialmente aspectos relacionados com a troca de documentos entre as aplicações, ou seja, entre a aplicação cliente e o Web Service, e também como os Web Services podem ser descobertos dinamicamente pelas aplicações cliente. Esses documentos têm vários objectivos: implementar a comunicação de interacção entre as aplicações, através do protocolo SOAP, outro documento, WSDL, permite descrever a interface do serviço, ou seja, os métodos que são disponibilizados pelo Web Service e também os parâmetros recebidos, a resposta a enviar e ainda o processo de comunicação com o servidor SOAP. A partir deste documento, ferramentas de *software* podem gerar código automaticamente para se conectar ao Web Service e usar as suas funcionalidades.

Para que os Web Services possam ser descobertos dinamicamente foi definido o UDDI, que permite, por um lado, aos fornecedores de Web Services publicitar o seus serviços e, por outro lado, aos clientes que precisam dos serviços desses Web Services, procurar o Web Service que melhor cumpre os requisitos das suas necessidades. O próprio registo UDDI baseia-se num Web Service.

Qualquer destes protocolos abordados anteriormente usam como linguagem o XML, ou seja, qualquer documento trocado entre as aplicações são elaborados segundo as regras do XML.

Para fornecer a comunicação, ou seja efectuar a troca de mensagens entre as aplicações, adoptou-se o protocolo HTTP, de ampla utilização, sendo hoje o principal mecanismo de comunicação na *Web*. Embora o HTTP seja o utilizado como norma, outros protocolos de transporte podem ser usados, como SMTP, FTP, filas de mensagens e mensagens instantâneas.

Para atender aos requisitos do modelo da Arquitectura Orientada a Serviços foram adoptadas as seguintes normas para cada uma das necessidades no desenvolvimento de aplicações baseado em serviços:

- Um formato normalizado de representar os dados – XML;
- Uma infra-estrutura de comunicação global – TCP/IP;
- Um protocolo de transporte de dados normalizado – HTTP;
- Um formato de mensagens comum – SOAP;
- Uma linguagem de descrição dos serviços – WSDL;
- Um modo de descobrir serviços e fornecedores de serviços na Web – UDDI.

Os protocolos que constituem a arquitectura básica dos Web Services e respectiva função podem ser vistos na figura 3.11.

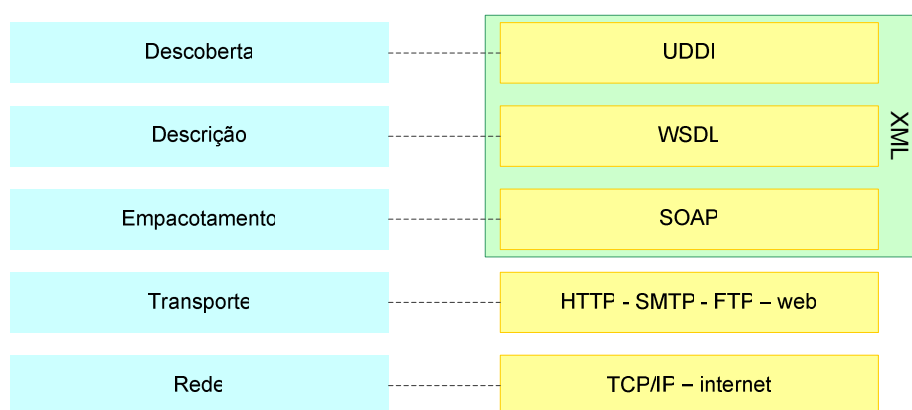


Figura 3.11 – Protocolos básicos dos Web Services

Os protocolos avançados definem aspectos relacionados com a segurança, gestão das transacções, coreografia e orquestração dos Web Services, e ainda aspectos relacionados com o encaminhamento de mensagens. Esses protocolos serão abordados na secção 3.6.

3.5 Protocolos básicos

O conceito de Web Services resulta de mais uma tentativa para permitir a interacção automática entre aplicações utilizando a *Web* como meio de conexão. Assim, qualquer especificação candidata a integrar a pilha de protocolos dos Web Services tem, obrigatoriamente, ter como alvo a interoperabilidade. Portanto, para a arquitectura básica não foi definido nenhum protocolo novo, mas sim definida a forma como utilizar os protocolos já existentes, para que possam cooperar a fim de atingir o mesmo objectivo.

Actualmente, os principais protocolos adoptados para a implementação básica de Web Services são os seguintes: HTTP, XML, SOAP, WSDL e UDDI.

3.5.1 HTTP

HTTP significa *HyperText Transfer Protocol*, e diz respeito a um protocolo que foi criado em 1990 com o objectivo de realizar a transferência de informações entre computadores. Apesar de, por definição, o protocolo HTTP poder ser usado em vários contextos, ganhou maturidade na *Web*, como principal mecanismo de comunicação.

As transferências de dados na *Web* baseiam-se numa arquitectura cliente-servidor, em que do lado do servidor está em execução um programa que espera pedidos oriundos do cliente. Esse programa é conhecido como servidor de requisições HTTP [W3C, 1996].

A figura 3.12 mostra o processo de comunicação entre cliente e servidor utilizando o protocolo HTTP.

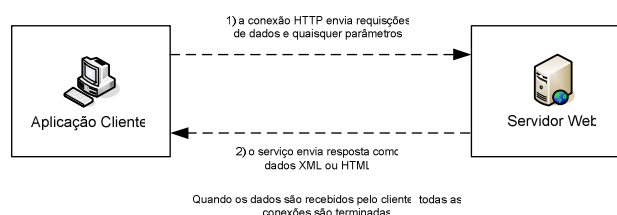


Figura 3.12 – Processo de transacção HTTP

Uma desvantagem reside no facto da conexão cliente-servidor não ser persistente. Portanto, em situações em que os clientes fazem um grande número de transacções distintas com o servidor, uma grande percentagem do tempo é gasto na criação e na terminação de conexões entre os clientes e o servidor. Para contornar este problema, é possível utilizar o mecanismo *keep-alive* da versão HTTP 1.1, em que a conexão entre o cliente e o servidor é mantida aberta durante as transacções [W3C, 1999] [IETF, 2000a]. Contudo, recorrendo a esta solução, o servidor fica limitado a um menor número de pedidos de requisição activos.

Outra característica, que pode ser considerada uma desvantagem, está ligada ao facto de os clientes ficarem bloqueados à espera da resposta do servidor. Este tipo de mecanismo leva a um comportamento síncrono, o que limita a escalabilidade das aplicações.

Inicialmente, o HTTP era o único protocolo usado pelos Web Services como meio de transporte e comunicação entre clientes e serviços. Actualmente é possível utilizar outros protocolos como por exemplo:

- Filas de mensagens;
- SMTP, *Simple Mail Transfer Protocol*;
- FTP, *File Transfer Protocol*;
- Mensagens instantâneas.

3.5.2 XML

A ideia de trocar dados entre sistemas de computação nasceu no mesmo dia em que foi instalado o segundo sistema. No entanto, por diversos motivos, essa troca nem sempre foi facilitada, acabando por se criarem muitas ilhas de dados.

Cada vez mais, os procedimentos computacionais actuais envolvem a combinação de dados de diferentes sistemas. A forma mais simples para o conseguir parece ser exportar os dados de um sistema para um formato legível e importá-los noutra sistema. O texto na sua forma mais básica, em ASCII, pode ser usado. No entanto, é necessário ter uma forma de descobrir o significado desse texto.

Estes conceitos estão na génese da linguagem XML que significa *eXtensible Markup Language*. Um documento XML é um documento constituído por *tags*, ou seja, marcas, que indicam o que os dados significam.

A XML foi definida em 1998 pelo W3C [W3C, 2005a], para representação de dados e contém um conjunto de regras de sintaxe fixas e obrigatórias. Essas regras determinam o formato que as *tags* devem ter, os caracteres permitidos e os seus significados, assim como o formato do documento em geral. Na figura 3.13 podemos observar a estrutura básica de um documento XML.

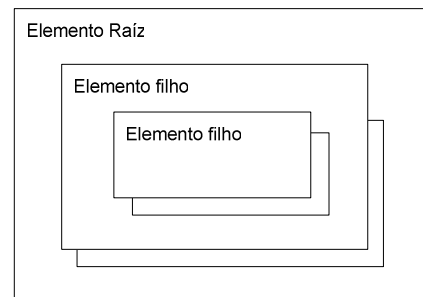


Figura 3.13 – Estrutura geral de um documento XML

A especificação XML não define quais as *tags*, ou seja, os elementos a usar, nem tão pouco o que significam. O vocabulário é criado pelo utilizador do XML, conforme as suas necessidades, ou seja, as *tags* a criar e o seu significado são da inteira responsabilidade do criador do documento.

Muitas vezes, o vocabulário de um documento XML surge de um acordo entre duas ou mais partes. Para cada contexto onde há a necessidade de troca de dados é normal surgir um vocabulário distinto. Nos pontos a seguir, apresentamos as principais características do XML:

- Separação do conteúdo e da formatação;
- Legível por humanos e por máquinas;
- Possibilidade de criação de *tags* sem limitação;
- Criação de documentos de validação de estrutura;
- Interligar bases de dados distintas.

Relativamente ao último ponto, uma base de dados pode ser exportada para um documento XML, permitindo assim, que o seu conteúdo possa ser processado por uma outra base de dados.

3.5.2.1 Componentes do XML

A especificação define um conjunto de componentes que se integram para suportar as aplicações que se baseiam ou tratam XML.

Esses componentes permitem criar sistemas que transfiram dados facilmente, que podem ser validados e interpretados e processados.

Os componentes principais que compõem o XML são:

- **Documento XML** – Um ficheiro que obedece às regras da norma XML. Contém dados, pode ser visto como uma base de dados.
- **Parser XML** – Programa de computador que interpreta o conteúdo do documento XML. O *parser* transforma o documento XML em estruturas de dados que podem facilmente ser processados.
- **Document Type Definition (DTD)** – Estrutura de *tags* que são permitidas num documento XML e as relações entre *tags*. Permite validar um documento XML. Entretanto, como não se trata de um documento XML, vai sendo substituída pela especificação do esquema XML.
- **Esquema XML** – Estrutura de *tags* que são permitidas num documento XML e as relações entre *tags*. O documento XML é validado comparando-o com o esquema XML definido.
- **Namespaces** – Um nome exclusivo deve ser usado para evitar conflitos entre nomes de *tags*. Como um documento XML poder conter outros documentos XML, é necessário garantir que nenhuma das *tags* destes documentos tenha o mesmo nome. Podemos então, criar um *namespace* para cada documento XML.

3.5.2.2 Regras do XML

O XML tem duas formas de estruturar os dados num documento:

- Elementos

São estruturas que permitem atribuir dados simples, tais como sequências de números e letras, ou seja elementos simples, e de outros elementos complexos.

Os nomes dos elementos são definidos entre os sinais “<” (menor) e “>” (maior) , a este conjunto dá-se o nome de *tag* ou nó. Cada *tag* que é aberta deve

ser fechada, o que é feito com a repetição da *tag* de abertura, porém, utilizando-se “/” (barra) antes do nome do elemento. Existe ainda, um tipo de *tags* que não contém mais dados que apenas o nome e atributos. Neste caso, antes do sinal de maior, deve ser colocada a barra.

O conteúdo do elemento é colocado entre as *tags* de abertura e a de fechamento.

- Atributos

São uma estrutura para definir informações auxiliares, metainformações e são colocados junto às *tags* de abertura dos elementos.

No anexo A, temos um ficheiro XML, com o próprio esquema XML integrado. Esse documento XML é usado para a aplicação wsQL fazer chegar os parâmetros à aplicação cliente do Web Service, depois de devidamente validada.

3.5.3 SOAP

Na secção anterior analisamos uma linguagem que possui um formato simples para representação de dados, que se tornou uma norma para troca de dados entre sistemas heterogéneos. Portanto, em termos de linguagem já temos uma que pode ser processada por qualquer sistema. Falta definir um formato de mensagem que não esteja vinculado a qualquer arquitectura de hardware ou *software*, mas que transmita uma mensagem de qualquer plataforma para uma outra qualquer plataforma de uma maneira clara e eficiente.

O formato com essas características está patente no protocolo SOAP, *Simple Object Access Protocol*, que é uma norma aberta que foi definida em 1999, como uma especificação do W3C, tendo como base uma proposta conjunta da Microsoft, IBM e Userland.com, entre outras empresas [W3C, 2005g].

SOAP é um protocolo baseado em XML, que realiza troca de dados, utilizando habitualmente como protocolo de transporte o HTTP, mas outros protocolos podem ser usados, como SMTP, FTP e filas de mensagens, como JMS e MSQ. Foi definido para invocar procedimentos remotos através do mecanismo RPC, *Remote Procedure Call*, ou para troca de mensagens, num ambiente independente da plataforma e da linguagem de programação.

SOAP é, portanto, uma escolha natural para se utilizar com os Web Services. Desta forma, pode-se garantir a interoperabilidade e intercomunicação entre diferentes sistemas,

através da utilização de uma linguagem normalizada, o XML e, um mecanismo de transporte normalizado, o HTTP.

Portanto, uma mensagem SOAP é um documento XML. Assim, a utilização do SOAP pode ser visto como um conjunto de camadas, como mostra a figura 3.14 [Potts, 2002]

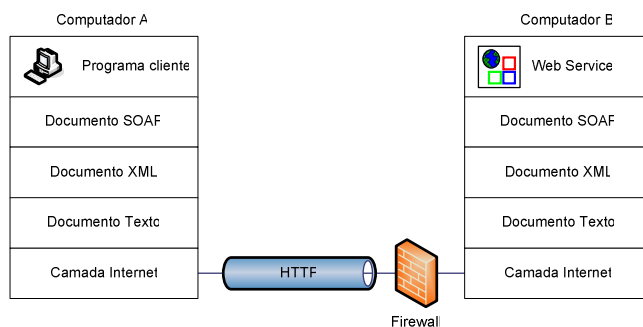


Figura 3.14 – SOAP visto em Camadas

Para a camada da Internet, o computador A está a enviar um documento XML para o computador B recorrendo ao protocolo HTTP. A política de segurança da *firewall* do computador B deixa passar documentos XML através de HTTP. O servidor *Web* do computador B recebe o documento e entrega-o ao processador SOAP, que usa um *parser* XML para ler o documento. O mecanismo SOAP examina o vocabulário SOAP, para determinar se é válido. Caso seja válido, o processador SOAP, chama o Web Service descrito no documento e passa-lhe os parâmetros que o documento contenha.

Quando o Web Service termina o processamento, cria uma resposta em XML, num formato específico da aplicação, que é empacotada numa mensagem SOAP, e entrega-a ao servidor *Web*, para este a devolver ao computador A.

No computador A, o programa cliente HTTP recebe o documento de resposta e chama o processador SOAP para analisar e validar o documento. Se for um documento válido, o processador SOAP passa o documento ao programa cliente que enviou a requisição ao Web Service do computador B.

Nos pontos seguintes evidenciamos as principais características do SOAP:

- Definido pelo consórcio W3C;
- Criado para invocar procedimentos remotos;
- Protocolo baseado em XML para troca de dados num ambiente distribuído e heterogéneo;
- Utilizado nos Web Services;
- Normalmente utiliza HTTP como protocolo de transporte.

3.5.3.1 Componentes do SOAP

O SOAP, na prática, é utilizado para descrever como chamar procedimentos num computador remoto. Essa descrição representa um acordo entre as partes envolvidas na troca de dados. Uma mensagem SOAP é composta por duas partes obrigatórias: o envelope SOAP e o corpo SOAP, e por uma parte opcional, o cabeçalho SOAP.

Todas as *tags* XML associadas ao SOAP possuem o prefixo *soap*, assim os principais elementos de uma mensagem SOAP são:

- soap:Envelope
- soap:Header
- soap:Body

A figura 3.15 mostra a relação entre as partes que constituem uma mensagem SOAP.

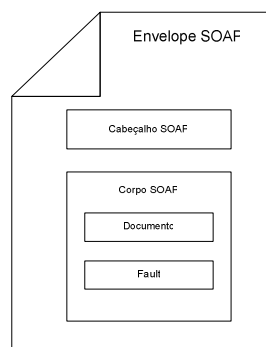


Figura 3.15 – Envelope SOAP

O exemplo 3.1 mostra a composição de uma mensagem SOAP contendo os dados para os parâmetros *Utilizador* e *Código*. Esta mensagem vai ser enviada através de uma requisição HTTP a um Web Service, para executar o método *Login* com os valores enviados para os parâmetros.

Exemplo 3.1 – Estrutura mensagem SOAP

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <Login xmlns="http://fe.up.pt/wsQL/wsQL">
      <Utilizador>JM</Utilizador>
      <Codigo>mtm</Codigo>
    </Login>
    <soap:Fault>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

3.5.3.1.1 Elemento soap:Envelope

O elemento raiz de uma mensagem SOAP é *soap:Envelope*. Contém elementos com *namespaces* globais que permitem evitar conflitos de nomes de elementos. A sintaxe do elemento *soap:Envelope* é mostrada no exemplo 3.2.

Exemplo 3.2 – Elemento soap:Envelope numa mensagem SOAP

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
. . .
</soap:Envelope>
```

Cada envelope encerra apenas uma mensagem, no entanto, pode conter vários elementos de cabeçalho. A string "http://schemas.xmlsoap.org/soap/envelope/" serve para indicar ao receptor da mensagem, apesar de uma forma indirecta, a versão do SOAP que foi utilizada para codificar a mensagem. Assim, o receptor pode facilmente determinar se é capaz de tratar a mensagem da versão indicada. Outros *namespaces* comuns numa mensagem SOAP são *xsd* e *xsi*, para respectivamente especificar se as *tags* são da definição do esquema XML ou são provenientes do *schema-instance XML*.

3.5.3.1.2 Elemento soap:Header

O elemento *soap:Header* é opcional numa mensagem SOAP. No caso de existir, será o primeiro elemento filho do elemento *soap:Envelope*. A especificação não define um formato em termos de elementos permitidos para o elemento *soap:Header*, portanto, tanto programas clientes como programas servidores podem usar este bloco do envelope SOAP para comunicar outros dados. Um uso típico consiste em usar o cabeçalho para comunicar as credenciais como o nome do utilizador e código de acesso.

No entanto, a especificação define dois atributos para o *soap:Header*, o *soap:mustUnderstand* e o *soap:actor*. Se *soap:mustUnderstand* estiver definido com o valor "1", indica que uma mensagem de erro deve ser gerada caso o programa receptor não estiver programado para tratar campos de cabeçalho. O atributo *soap:actor* serve para indicar quais os programas receptores a que a mensagem se destina, ou seja, por onde tem que passar para completar o processamento pretendido. Esse percurso da mensagem pode

ser criado acrescentando ao cabeçalho as *tag soap:actor* com os respectivos URIs. No exemplo 3.3 temos uma mensagem SOAP com um cabeçalho simples.

Exemplo 3.3 – Elemento soap:Header numa mensagem SOAP

```

. . .
<soap:Header>
  <Autenticacao soap:mustUnderstand = "1">
    <LoginID>feup</LoginID>
    <Password>mtm</Password>
  </Autenticacao>
</soap:Header>
. . .

```

Este cabeçalho contém um elemento `<Autenticacao>` contendo os elementos `<LoginID>` e `<Password>` que servem para comunicar as credenciais a serem usadas no destino. Espera-se que futuramente esta área da mensagem SOAP seja especificada para suportar outros recursos como sessões, transacções e autenticação de utilizadores.

3.5.3.1.3 Elemento soap:Body

Este elemento define o início do corpo da mensagem. Este elemento é obrigatório e contém o *payload*, ou seja, a informação a ser transportada até ao destino final. O *payload* pode ser uma chamada de invocação de um método num computador remoto ou apenas um documento XML que é necessário fazer chegar a um destino ou ainda uma mensagem de resposta. No exemplo 3.4 temos um fragmento de uma mensagem SOAP que faz uma chamada de procedimento remoto a um método chamado *Login*.

Exemplo 3.4 – Elemento soap:Body numa mensagem SOAP

```

. . .
<soap:Body>
  <Login xmlns="http://fe.up.pt/wsQL/wsQL">
    <Utilizador>JM</Utilizador>
    <Codigo>mtm</Codigo>
  </Login>
  . . .
</soap:Body>
. . .

```

O nome do método a ser chamado é usado para formar o nome do elemento no corpo da mensagem, no exemplo, o elemento `<Login>`. Os dois elementos filhos deste elemento, `<Utilizador>` e `<Codigo>` representam os parâmetros que o método requer para ser executado.

3.5.3.1.4 Elemento soap:Fault

Num ambiente distribuído e heterogéneo podem ocorrer vários erros quando se invoca uma método remoto, exemplos de erros são, métodos inexistentes ou falta de parâmetros. No caso em que não podem ser corrigidos no computador remoto, o computador cliente deve receber uma notificação da falha com as informações necessárias à correcção.

Para tratar erros no processamento das mensagens, a especificação SOAP definiu o elemento *soap:Fault* que é um elemento filho de *soap:Body*. Este elemento opcional é necessário apenas nas mensagens de resposta. O elemento *soap:Fault* possui quatro elementos:

- **soap:faultcode** – elemento exigido pela especificação. Contem um código computacional que indica qual é o problema. Quatro tipos de *faultcode* estão definidos:
 - server** – erro ocorreu no servidor, o cliente terá que rescrever o código que invoca o método.
 - client** – erro do lado do cliente, por exemplo, formato da mensagem, mensagem incompleta, etc..
 - versionMismatch** – quando as versões do SOAP do cliente e servidor são diferentes.
 - mustUnderstand** – este erro é gerado quando um elemento do cabeçalho não pode ser processado.
- **soap:faultstring** – texto que descreve o erro representado pelo código indicado no elemento *soap:faultcode*.
- **soap:faultactor** – este elemento opcional informa qual o método responsável pela falha, é essencial quando vários métodos são chamados.
- **soap:detail** – este elemento deve conter o máximo de informação acerca do estado do servidor no momento da falha. Por exemplo, os valores das variáveis no momento da falha.

No exemplo 3.5 temos um exemplo do elemento para tratar falhas num mensagem SOAP.

Exemplo 3.5 – Elemento soap:Fault numa mensagem SOAP

```

. . .
<soap:Body>
. . .
  <soap:Fault>
    <soap:faultcode>Client.Authentication</soap:faultcode>
    <soap:faultstring>O utilizador não existe no nosso sistema</soap:faultstring>
    <soap:faultactor>http://fe.up.pt/wsQL/wsQL.asmx/login</soap:faultactor>
    <soap:detail>
      <Utilizador>JM</Utilizador>
    </soap:detail>
  </soap:Fault>
</soap:Body>
. . .

```

De notar que o elemento *soap:faultcode* possui um formato específico e que o elemento *soap:faultactor* contem o URI de um Web Service. Os outros dois elementos podem conter qualquer informação útil para a comunicação.

3.5.3.2 Regras de codificação SOAP

Um dos problemas patentes em comunicações de dados entre sistemas computacionais diz respeito à representação dos tipos de dados. A declaração dos tipos de dados é fundamental para que um programa de computador funcione correctamente.

O SOAP permite colocar dados na mensagem sem definir o tipo de dados, nesse caso, os dados serão considerado do tipo padrão *string*. Entretanto, podemos definir um tipo de dados colocando um atributo junto ao elemento, como mostrado no exemplo:

```
<Nota xsi:type="xsd:int">16</Nota>
```

O SOAP tira partido do facto de todos os sistemas computacionais existentes poderem processar dados num formato básico: texto.

A tarefa de conversão de tipos de dados é da responsabilidade dos programas servidores e dos programas clientes. A figura 3.16 mostra graficamente a conversão de *string* em *número* e de *número* em *string*.

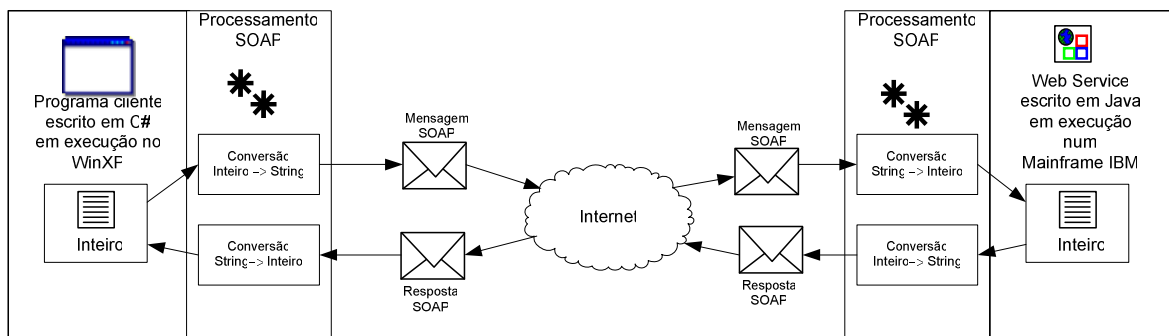


Figura 3.16 – Conversão de dados: *string*↔*Inteiro*

Outra maneira de indicar o tipo de dados de um elemento é usando o esquema XML. No exemplo 3.6 temos um esquema XML, ou seja, o XSD, *XML Schema Data*, integrado numa mensagem SOAP.

Exemplo 3.6 – Esquema XML numa mensagem SOAP

```

. . .
    <xs:element name="Parametros">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="IDutilizador" type="xs:string" minOccurs="0" />
          <xs:element name="Nome" type="xs:string" minOccurs="0" />
          <xs:element name="Empresa" type="xs:string" minOccurs="0" />
          <xs:element name="UltimoAcesso" type="xs:dateTime" minOccurs="0" />
          <xs:element name="IDbd" type="xs:int" minOccurs="0" />
          <xs:element name="BaseDados" type="xs:string" minOccurs="0" />
          <xs:element name="Tabela" type="xs:string" minOccurs="0" />
          <xs:element name="Seleccionar" type="xs:boolean" minOccurs="0" />
          <xs:element name="Inserir" type="xs:boolean" minOccurs="0" />
          <xs:element name="Actualizar" type="xs:boolean" minOccurs="0" />
          <xs:element name="Apagar" type="xs:boolean" minOccurs="0" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
. . .

```

Portanto, as regras de codificação SOAP têm por objectivo orientar a conversão dos dados que estão originalmente representados numa linguagem de programação para o formato XML de forma a seguir o formato do protocolo SOAP. Esta conversão permite que os objectos sejam transportados via Internet utilizando o protocolo HTTP ou outro.

3.5.3.3 SOAP com HTTP

O protocolo SOAP pode ser implementado para ser usados com vários protocolos de transporte. Porém, o corrente é implementa-lo em associação com o protocolo HTTP. Por este facto, e também por este ser o protocolo escolhido para implementar o trabalho de suporte a esta dissertação, vamos apresentar aspectos sobre a relação do SOAP com HTTP.

O HTTP é um protocolo para realizar a transferência de informações entre computadores. Associado aos Web Services, o protocolo HTTP inclui nas suas mensagens o conteúdo da mensagem SOAP. Um exemplo do uso do protocolo HTTP para transportar uma requisição SOAP é apresentado no exemplo 3.7.

Exemplo 3.7 – Requisição SOAP embebida em HTTP

```

1  POST /wsQL/wsQL.asmx HTTP/1.1
2  Host: localhost
3  Content-Type: text/xml; charset=utf-8
4  Content-Length: 512
5  SOAPAction: "http://fe.up.pt/wsQL/wsQL/Login"
6
7  <?xml version="1.0" encoding="utf-8"?>
8  <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
11 <soap:Body>
12   <Login xmlns="http://fe.up.pt/wsQL/wsQL">
13     <Utilizador>feup</Utilizador>
14     <Codigo>mtm</Codigo>
15   </Login>
16 </soap:Body>
17 </soap:Envelope>

```

Nas linhas de 1 a 5 está definido o cabeçalho HTTP, onde é possível identificar que esta é uma mensagem de requisição através do comando *POST*. Na primeira linha está definido o recurso para o qual a mensagem é destinada e a versão do protocolo HTTP. A segunda linha identifica o nome do *host* de destino. A terceira linha indica o tipo de conteúdo que segundo a especificação SOAP é obrigatório que seja igual a “*text/xml*”, esta linha define ainda o tipo de codificação de caracteres, no caso, “*utf-8*”. A linha 4 informa o tamanho do conteúdo da mensagem. A quinta linha com a directiva *SOAPAction* define o método a ser invocado no computador remoto. As linhas 12 a 15 indicam o método a ser invocado e os valores para os parâmetros exigidos. Uma resposta a esta requisição é apresentada no exemplo 3.8.

Exemplo 3.8 – Resposta SOAP embebida em HTTP

```

1  HTTP/1.1 200 OK
2  Content-Type: text/xml; charset=utf-8
3  Content-Length: 1512
4
5  <?xml version="1.0" encoding="utf-8"?>
6  <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
9  <soap:Body>
10   <LoginResponse xmlns="http://fe.up.pt/wsQL/wsQL">
11     <LoginResult>
12       <NewDataSet>
13         <Parametros>
14           <IDutilizador>OZG3rhG51xnRw1E6cbvT</IDutilizador>
15           <Nome>JM</Nome>
16           <Empresa>mtm:FEUP</Empresa>
17           <UltimoAcesso>2004-11-17T02:24:00</UltimoAcesso>
18         </Parametros>
19         . . .
20       </NewDataSet>
21     </LoginResult>
22   </LoginResponse>
23 </soap:Body>
24 </soap:Envelope>

```

Na resposta, o cabeçalho HTTP contém menos informações porque cada resposta é relacionada pelo protocolo à sua respectiva requisição. Assim, na mensagem de resposta não existem os campos *host* nem *SOAPAction*. Os dados referentes ao retorno que o método invocado devolve à aplicação cliente encontram-se nas linhas 12 a 20.

3.5.4 WSDL

Uma das grandes vantagens dos Web Services, em relação às tecnologias precedentes, é forma como os serviços são descritos. Essa descrição fornece todas as informações necessárias para um cliente se ligar ao Web Service, assim como todos as funções disponíveis, respectivos parâmetros e tipo de dados. Além disso, contém alguma informação que indica o objectivo do Web Service.

Qualquer aplicação que utilize um determinado Web Service é programada a partir do conhecimento extraído deste documento de descrição. Obviamente, antes que as comunicações via Web Services sejam iniciadas, é necessário que a máquina que vai comunicar com o Web Service tenha conhecimento dos formatos de mensagens que serão recebidas e enviadas pelo servidor. Para este fim, foi criada o WSDL, *Web Service Description Language* [W3C, 2005c], que é uma linguagem baseada em XML para a descrição das funcionalidades de um Web Service.

As informações do documento WSDL são muito úteis uma vez que, devido à linguagem utilizada, são facilmente interpretados por humanos e várias ferramentas de *software* sabem gerar código automaticamente para se conectar fisicamente ao Web Service e utilizar as suas operações.

3.5.4.1 Elementos do documento WSDL

A estrutura de um documento WSDL é subdividida logicamente em dois blocos: as descrições abstractas e as descrições concretas, ou então, as descrições não funcionais e as descrições funcionais. Os quatro elementos XML abstractos que podem ser definidos num documento WSDL são os seguintes:

- `<wsdl:types>`
- `<wsdl:message>`
- `<wsdl:portType>`
- `<wsdl:operation>`

Os elementos abstractos são utilizados para descrever os dados e as operações suportadas pelo Web Service. Podem ser definidos também quatro elementos XML concretos num documento WSDL:

- `<wsdl:service>`
- `<wsdl:port>`
- `<wsdl:binding>`
- `<wsdl:operation>`

Os elementos concretos descrevem aspectos físicos, ou seja, como o cliente se deve conectar ao Web Service pretendido. A descrição concreta está para os Web Services como o IDL está para o CORBA.

Todos estes elementos, abstractos e concretos, são colocados como elementos filhos do elemento raiz, *definitions*, seguindo a ordem apresentada. Além destes elementos, um documento WSDL poderá conter mensagens SOAP e outras definições XML. Na figura 3.17 podemos ver a estrutura de um documento WSDL. No anexo D, temos um documento WSDL completo, que descreve o Web Service de suporte a este trabalho, o wsQL.

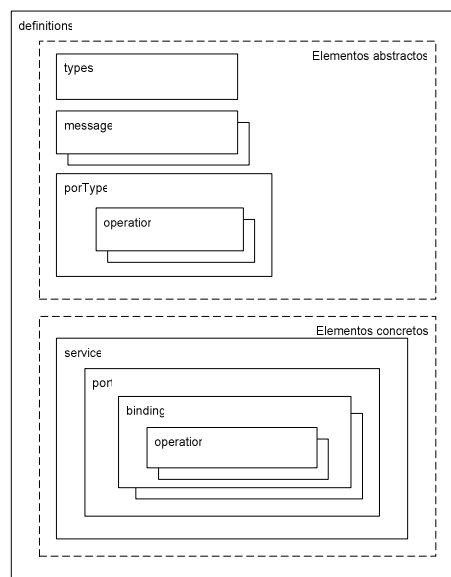


Figura 3.17 – Elementos do documento WSDL

Elemento *definitions*

O elemento raiz de um documento WSDL é *definitions*. Contém elementos para especificar o *targetNamespace* bem como outros *namespaces* globais que permitem evitar conflitos de nomes de elementos. A sintaxe do elemento *definitions* pode ser analisada no exemplo 3.9.

Exemplo 3.9 – Elemento *definitions* no documento WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://fe.up.pt/wsQL/wsQL"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://fe.up.pt/wsQL/wsQL" xmlns="http://schemas.xmlsoap.org/wsdl/">
.
.
.
</definitions>
```

Elemento *types*

O elemento *types* é responsável por definir um esquema para os dados que serão trocados nas requisições e nas respostas dos métodos do Web Service. A linguagem normalizada utilizada para a representação deste esquema é o XML Schema.

Apesar da especificação SOAP numa das suas regras impor que seja apenas permitido um valor de entrada e um valor de saída para mensagens enviadas pela Internet, esse valor pode ser um tipo complexo, como por exemplo, um *array*.

Um exemplo da utilização deste elemento num documento WSDL é apresentado no exemplo 3.10.

Exemplo 3.10 – Elemento *types* no documento WSDL

```
.
.
.
<types>
<s:schema elementFormDefault="qualified" targetNamespace="http://fe.up.pt/wsQL/wsQL">
  <s:import namespace="http://www.w3.org/2001/XMLSchema" />
  <s:element name="Login">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Utilizador" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Codigo" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="LoginResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="LoginResult">
          <s:complexType>
            <s:sequence>
```

```

        <s:element ref="s:schema" />
        <s:any />
      </s:sequence>
    </s:complexType>
  </s:element>
</s:sequence>
</s:complexType>
</s:element>
. . .
</types>
. . .

```

Elemento *message*

Os elementos *message* definem os formatos abstratos para os dados que são enviados e recibos na comunicação entre os dois intervenientes do Web Service. O valor do atributo *name* de cada elemento *message* possui o sufixo *In* ou *Out* consoante se trate respectivamente de dados enviados ou recibos.

Basicamente, os elementos *message* representam os parâmetros dos métodos, comumente chamados de operações em termos de WSDL.

Estes elementos são compostos por elementos *part* nos quais é estabelecida uma relação com a secção *types* através do atributo *element*. A utilização deste elemento pode ser analisado no exemplo 3.11.

Exemplo 3.11 – Elemento *messages* no documento WSDL

```

. . .
<message name="LoginSoapIn">
  <part name="parameters" element="s0:Login" />
</message>
<message name="LoginSoapOut">
  <part name="parameters" element="s0:LoginResponse" />
</message>
. . .

```

Elemento *operation*

Com os elementos *operation* são definidas as chamadas a métodos de forma análoga à chamadas de métodos ou funções numa linguagem de programação. Numa operação são permitidas três tipos de mensagens:

- **Mensagem de entrada** – define os dados que o Web Service está preparado para receber.
- **Mensagem de saída** – define os dados que o Web Service pode enviar como resposta.

- **Mensagem de falha** – define as mensagens de erro que podem ser retornadas pelo Web Service.

Em termos de operações possíveis num documento WSDL podem ser definidos os seguintes tipos de operações:

- **Requisição/Resposta** – um cliente faz uma requisição e o Web Service responde.
- **Solicitação/Resposta** – um Web Service envia uma mensagem ao cliente e o cliente responde.
- **Sentido único** – um cliente envia uma mensagem ao Web Service mas não espera uma resposta.
- **Notificação** – um Web Service envia uma mensagem a um cliente mas não espera resposta.

A sintaxe de uma operação é simples. Se a operação for do tipo requisição/resposta, o formato será como o mostrado no exemplo 3.12.

Exemplo 3.12 – Elemento *operation* no documento WSDL

```
. . .
<operation name="Login">
  <input message="s0:LoginSoapIn" />
  <output message="s0:LoginSoapOut" />
</operation>
. . .
```

Caso a operação seja do tipo solicitação/resposta, o elemento *output* será definido antes do elemento *input*. Se a operação for do tipo sentido único, o elemento *output* é omitido. Se a operação for do tipo notificação, o elemento *input* é omitido.

Elemento *portType*

Com os elementos *portType* define-se o conjunto de todas as operações que um determinado Web Service pode aceitar. *portType* é um elemento abstracto que não fornece informações sobre como o cliente deve-se conectar directamente a um Web Service. Este elemento fornece um bloco de informação do qual o cliente pode obter conhecimento sobre todas as operações que um Web Service oferece.

Estes elementos relacionam os tipos de mensagens definidos anteriormente nos elementos *message* através de atributos *message*.

Na versão 2.0 da especificação WSDL os elementos *porttype* foram renomeados para *interfaces* [W3C, 2005d]. O exemplo 3.13 apresenta um exemplo da utilização deste elemento.

Exemplo 3.13 – Elemento *portType* no documento WSDL

```

. . .
<portType name="wsQLSoap">
  <operation name="Login">
    <input message="s0:LoginSoapIn" />
    <output message="s0:LoginSoapOut" />
  </operation>
  <operation name="FechaSessao">
    <input message="s0:FechaSessaoSoapIn" />
    <output message="s0:FechaSessaoSoapOut" />
  </operation>
  <operation name="GetEstruturaTabela">
    <input message="s0:GetEstruturaTabelaSoapIn" />
    <output message="s0:GetEstruturaTabelaSoapOut" />
  </operation>
  <operation name="GetDadosTabela">
    <input message="s0:GetDadosTabelaSoapIn" />
    <output message="s0:GetDadosTabelaSoapOut" />
  </operation>
  <operation name="ExecutaSQL">
    <input message="s0:ExecutaSQLSoapIn" />
    <output message="s0:ExecutaSQLSoapOut" />
  </operation>
  <operation name="EscreveDadosTabela">
    <input message="s0:EscreveDadosTabelaSoapIn" />
    <output message="s0:EscreveDadosTabelaSoapOut" />
  </operation>
  <operation name="HistoricoLigacoes">
    <input message="s0:HistoricoLigacoesSoapIn" />
    <output message="s0:HistoricoLigacoesSoapOut" />
  </operation>
</portType>

```

Elemento *binding*

Os elementos que abordamos até este momento são utilizados para especificar o que um Web Service é capaz de fornecer em termos de funcionalidades. Este elemento que agora abordamos e os seguintes servem para descrever as informações necessárias para que uma aplicação cliente possa ligar-se a um Web Service.

O elemento *binding* possui duas funções. A primeira, funciona como um vínculo entre os elementos abstractos e os elementos concretos no documento WSDL. A segunda função do elemento *binding* é fornecer um bloco para informações como o protocolo e o endereço do Web Service. A sintaxe para um elemento *binding* está mostrada no exemplo 3.14.

Exemplo 3.14 – Elemento *binding* no documento WSDL

```

. . .
<binding name="wsQLSoap" type="s0:wsQLSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <operation name="Login">
    <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/Login" style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
. . .
</binding>
. . .

```

A primeira linha contém o *binding* com o *portType* que criamos no ponto anterior. O mesmo *portType* pode aparecer em vários elementos *binding*. Podemos definir um *binding* para outro tipo de transporte, por exemplo o SMTP.

O *binding* do exemplo é definido para ser um *binding* SOAP que utilizará o protocolo HTTP para enviar as mensagens.

O elemento *soap:body* fornece detalhes sobre como a operação deve ser codificada. No exemplo apresentado não existe qualquer codificação indicado pelo atributo *use* do elemento *soap:body*.

Elemento *port*

Para que de facto o cliente possa ligar-se ao Web Service falta definir o endereço URL e a porta de acesso. Essa informação é colocada no elemento *port*. A sintaxe do elemento *port* é mostrada no exemplo 3.15.

Exemplo 3.15– Elemento *port* no documento WSDL

```

. . .
<port name="wsQLSoap" binding="s0:wsQLSoap">
  <soap:address location="http://localhost/wsQL/wsQL.asmx" />
</port>
. . .

```

Elemento *service*

No elemento *service* são referenciados todos os elementos *port*, os quais relacionam-se com elementos *binding* previamente definidos e definem o endereço lógico do Web Service. O exemplo 3.16 apresenta um exemplo de utilização do elemento *service*.

Exemplo 3.16– Elemento *service* no documento WSDL

```
. . .  
<service name="wsQL">  
  <port name="wsQLSoap" binding="s0:wsQLSoap">  
    <soap:address location="http://localhost/wsQL/wsQL.asmx" />  
  </port>  
</service>  
. . .
```

3.5.4.2 O documento WSDL

Quando analisado na sua totalidade, um documento WSDL representa uma descrição completa do Web Service. Inclui todas as informações necessárias para ligar uma aplicação cliente a um determinado Web Service.

Em resumo, a secção *types* define todas as estruturas de dados que serão utilizadas pelo Web Service. Os elementos *message* definem os formatos dos dados que vão ser trocados entre cliente e servidor, ou seja, dos parâmetros e dos valores retornados, através da utilização das estruturas definidas no elemento *types*. Os elementos *portTypes* definem as estruturas dos métodos, por meio dos elementos *operation*, que definem os nomes dos métodos, além de reunirem os tipos de mensagens entre cliente e servidor, através dos parâmetros definidos nos elementos *message*. As secções *binding* estabelecem a ligação entre as estruturas dos métodos, representadas pelos elementos *portType*, com as acções concretas que o Web Service suporta, definidas pelos elementos *soap:operation* e *http:operation*. Por fim, o elemento *service* define a localização concreta do Web Service, além de estabelecer ligações com as operações representadas pelos elementos *binding*.

3.5.5 UDDI

Outra das capacidades da arquitectura dos Web Services é o conceito de existir um registo universal de Web Services. Esta funcionalidade ajudará, por um lado, os fornecedores de Web Services a publicitar os seus serviços e, por outro lado, os clientes, que precisam dos serviços desses Web Services, a procurar o Web Service que melhor cumpre os requisitos das suas necessidades.

De forma a unir estas duas vontades, um consórcio de empresas definiu o UDDI, *Universal Discovery, Description and Integration* [UDDI, 2004]. As empresas que se

associam neste consórcio acendem às centenas; no entanto, o facto da Microsoft e a Sun estarem presentes nesta normalização, é quase sinónimo de sucesso já que toda a comunidade tende a acompanhar.

3.5.5.1 Arquitectura UDDI

O UDDI consiste num conjunto de directórios públicos, que podem ser mantidos por qualquer empresa. Todos os directórios públicos replicam as informações publicadas dos outros. Portanto, o UDDI propriamente dito pode ser visto como uma colecção replicada de Web Services, o que garante que podemos aceder a todos os Web Services acedendo apenas a um directório público. O UDDI pode ser descrito da seguinte forma:

- É um mecanismo normalizado e transparente para descrever serviços;
- Descreve métodos simples para solicitar o serviço;
- Especifica um registo central dos serviços acessível.

Alguns exemplos de directórios públicos são:

- <http://uddi.ibm.com>
- <http://uddi.microsoft.com>
- <http://uddi.sap.com>
- <http://uddi.hp.com>

Para se obter localizações de fornecedores de registo, pode-se visitar o sítio Web do UDDI – <http://www.uddi.org>, e escolher o desejado. Na figura 3.18 podemos analisar a arquitectura UDDI.

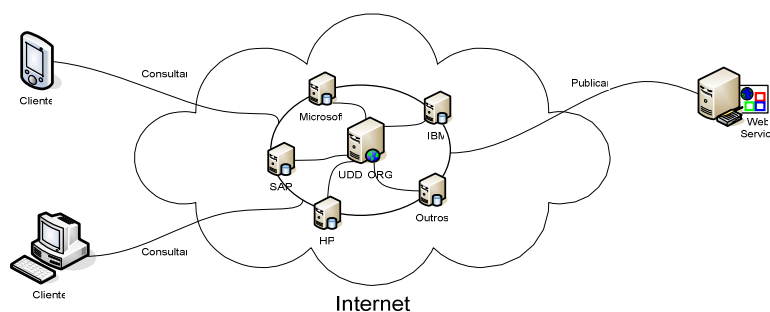


Figura 3.18 – Arquitectura UDDI

Os registos do UDDI são baseados num modelo confederado pelo que copiam os registos uns dos outros. Os registos podem estar fisicamente distribuídos; no entanto, estão

logicamente centralizados já que todos os registos são sincronizados periodicamente. As informações num registo UDDI são classificadas em três categorias principais:

- **Páginas Brancas** – incluem informação geral sobre uma empresa específica, como por exemplo, nome do negócio, descrição do negócio, informação de contacto, endereço, números de telefone, fax, ou mesmo incluir identificadores de negócios, no formato de classificação DUNS - *Data Universal Numbering System*, que são números de nove dígitos atribuídos a negócios. O UDDI versão 2.0 oferece suporte para identificadores específicos de indústrias, tal como o sistema SIC, *Standard Industrial Classification*, o qual atribui identificadores numéricos únicos a indústrias. Por exemplo, 7371 representa Serviços de Programação de Computadores.
- **Páginas Amarelas** – incluem dados de classificação geral para qualquer empresa ou serviço oferecido. Por exemplo, esses dados podem incluir a indústria, o produto, ou códigos geográficos baseados na taxionomia normalizada.
- **Páginas Verdes** – esta categoria contém informação técnica sobre um serviço na *Web*. Geralmente, essa informação inclui um apontador para uma especificação externa e um endereço para invocar o serviço. O UDDI não é restrito a descobrir serviços baseados em SOAP: ao contrário, pode ser usado também, para descrever qualquer serviço, desde uma única página *Web* ou endereços de *email*, até serviços CORBA, DCOM e JavaRMI.

Outra característica UDDI é o facto de ser construído sobre as mesmas normas SOAP dos Web Services. Portanto, um registo pode ser acedido para publicação ou consulta, através de qualquer linguagem de programação, por qualquer plataforma de hardware executando qualquer sistema operativo. Os registos UDDI funcionam exactamente como qualquer Web Service, como é mostrado na figura 3.19.

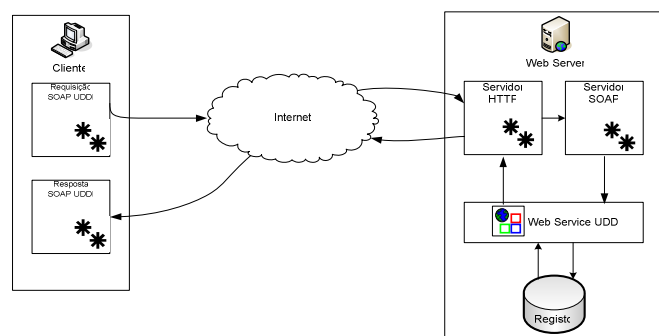


Figura 3.19 – O registo UDDI baseia-se num Web Service

3.5.5.2 Programação com UDDI

Todas as APIs da especificação UDDI são definidas em XML, inseridas em envelopes SOAP e enviadas por HTTP. Portanto, programar com as APIs do UDDI é análogo a programar com um qualquer Web Service.

Em teoria, conhecendo o URL do Web Service, qualquer aplicação se pode ligar a esse Web Service. No entanto, alguns Web Services podem ser definidos para receber documentos XML em formatos específicos.

Uma parte vital de toda a estratégia de busca UDDI é o seu modelo técnico, conhecido como *tModel*. O *tModel* fornece metadados sobre especificações adicionais, como o nome, o publicador do serviço e o URL do esquema XML. O *tModel* permite ao Web Service processar correctamente todas as mensagens que são enviadas nesse formato.

Basicamente, duas entidades interagem com o UDDI: a primeira, o fornecedor do Web Service, para publicar o serviço, ficando assim disponível para ser descoberto e utilizado; a segunda, o potencial cliente do serviço, que consulta o UDDI para pesquisar o Web Service adequado.

Esta pesquisa, ou seja a descoberta do Web Service, pode ocorrer de duas formas. A primeira, descoberta em tempo de projecto, envolve intervenção humana e programação. A segunda, descoberta em tempo de execução, é totalmente automática, isto é, as aplicações clientes descobrem e ligam-se automaticamente aos Web Services; no entanto, esta abordagem bastante avançada, envolve programação complexa.

Tudo indica que o UDDI possui um futuro brilhante. Brent Sleeper afirmou: “*o UDDI terá êxito porque a sua sustentação técnica funciona para os chatos...*” [Sleeper, 2001].

3.6 Protocolos avançados

Como referido anteriormente na secção 3.5, a arquitectura básica dos Web Services define um conjunto de protocolos normalizados para a construção de aplicações distribuídas servindo-se da infra-estrutura que Internet oferece, tendo sempre por objectivo principal a interoperabilidade. Esses protocolos tratam essencialmente de aspectos relacionados com a troca de documentos que activam a interacção automática entre a

aplicação cliente e o Web Service; portanto, esse conjunto de protocolos por si só são suficientes para construir aplicações baseadas em Web Services. Todavia, outros aspectos podem/devem ser considerados quando projectamos um Web Service, aspectos relacionados com a segurança, gestão de transacções, encaminhamento de mensagens e, ainda, aspectos relacionados com a coreografia e orquestração dos Web Services.

Este aspectos não foram tidos em conta na criação do protocolo SOAP porque, para cumprir os requisitos de independência de plataforma e linguagem de programação, foi necessário fazer algumas cedências ao nível da complexidade do protocolo, o que o tornou limitado a nível de, segurança, suporte a transacções, reencaminhamento e qualidade de serviço. Para fazer face a essas limitações as organizações que publicam normas estão actualmente a desenvolver especificações nesse sentido. Essas especificações serão abordadas nas secções seguintes.

3.6.1 Segurança

Uma vez que os Web Services utilizam para comunicar a infra-estrutura que a Web oferece, ficam inerentemente sujeitos aos problemas ao nível de segurança de que esta sofre. Relativamente ao tópico de segurança, está patente no senso comum a seguinte apreensão: as transacções via *Web* não são completamente seguras.

Evidentemente, sem garantias de segurança por um lado, os Web Services, nunca alcançarão o seu potencial máximo e, por outro lado, não conquistarão a confiança dos utilizadores de modo a serem utilizados em grande escala.

Basicamente, quando dois ou mais parceiros numa comunicação, humanos ou computacionais, efectuem uma transacção via *Web*, o que é necessário ao nível de segurança são mecanismos capazes de garantir os seguintes pontos:

- **Autenticidade** – ter a certeza de que a transacção ocorreu entre os dois parceiros pretendidos e não com terceiros simulando ser os verdadeiros;
- **Privacidade** – ter a certeza de que as mensagens trocadas entre os parceiros na comunicação não foram lidas por terceiros não autorizados;
- **Integridade** – ter a certeza de que as mensagens trocadas entre os parceiros na comunicação não foram alteradas;
- **Não rejeição** – todos os parceiros envolvidos na comunicação devem poder provar que são quem dizem ser, para não haver lugar à rejeição.

Por outro lado, como analisámos nas secções anteriores, toda a comunicação para interacção entre a aplicação cliente e o Web Service se processa através de mensagens SOAP. O protocolo SOAP é baseado em XML, ou seja em documentos de texto claro, perfeitamente legível.

Portanto, para tornar um Web Service seguro, dentro das limitações impostas pela *Web*, podemos encriptar a comunicação entre a aplicação cliente e o Web Service. Essa segurança pode-se concretizar de duas formas que de seguida vão ser abordadas.

3.6.1.1 Segurança a nível do transporte

A este nível, a segurança implementada tem por objectivo proteger o protocolo de rede que o Web Service utiliza para efectuar a transferência de mensagens entre a aplicação cliente e o Web Service. Este nível de segurança está ilustrado na figura 3.20 [Microsoft, 2002a].

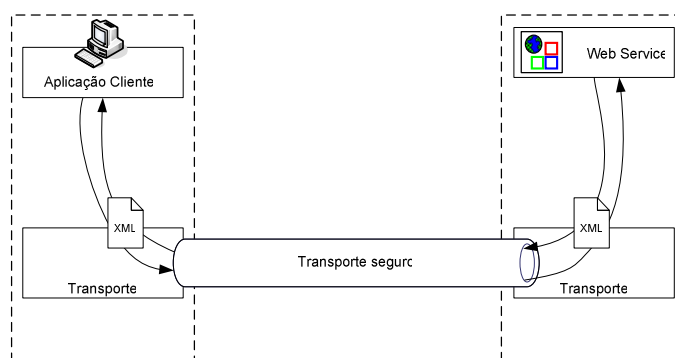


Figura 3.20 – Transporte seguro

Para fornecer um transporte seguro para o protocolo HTTP, o protocolo normalizado adoptado foi o SSL, *Secure Sockets Layer* [IETF, 2000b]. Com a implementação do SSL, a segurança é garantida através da encriptação de toda a comunicação entre a aplicação cliente e o Web Service. Neste modelo de segurança, a aplicação cliente abre um *socket* para o Web Service, que será usado para trocar mensagens seguras, ou seja mensagens SOAP utilizando o protocolo HTTPS.

A implementação de uma comunicação segura entre aplicação cliente e o Web Service recorrendo ao protocolo SSL pode ser observada na figura 3.21 [Microsoft, 2002a].

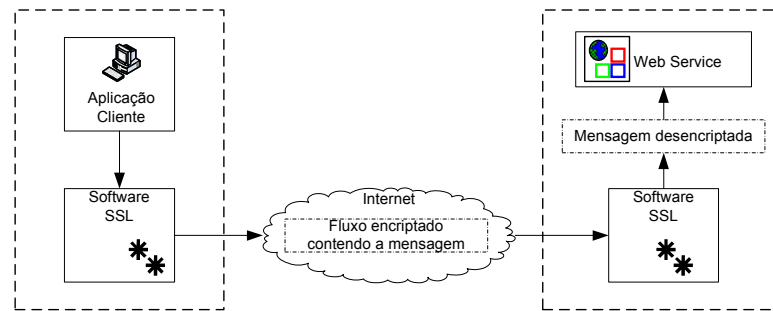


Figura 3.21 – Comunicação segura com SSL

Efectivamente, o protocolo SSL é a implementação mais comum para fornecer a comunicação segura entre um cliente e um servidor.

Uma alternativa é o IPSec. O projecto IPSec, desenvolvido pelo IETF, representa um esforço para desenvolver uma arquitectura de segurança para o protocolo IP para integrar mecanismos de autenticação, gestão e distribuição de chaves. O IPSec permite criar redes virtuais privadas e seguras através de redes públicas como a Internet.

3.6.1.2 Segurança a nível da mensagem

O SSL garante a protecção da mensagem durante o transporte; no entanto, como certas transacções de Web Services precisam ser encaminhadas e processadas por vários nós, isso significa que cada nó procede à encriptação e desencriptação, expondo desta forma a mensagem em cada nó.

Portanto, torna-se necessário um esquema de segurança que permita proteger o conteúdo da mensagem e, no caso de a mesma ser encaminhada e processada por vários nós, um nó tenha acesso apenas ao conteúdo que lhe é destinado.

A este nível, a segurança implementada envolve a encriptação dos documentos trocados pela aplicação cliente e pelo Web Service. Este nível de segurança está ilustrado na figura 3.22 [Microsoft, 2002a].

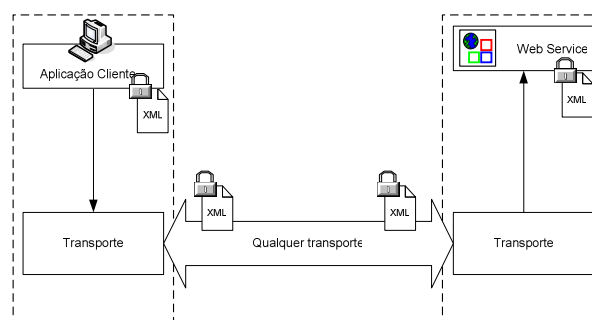


Figura 3.22 – Mensagem segura

A criptografia é a arte de escrita secreta, como tal torna-se a base dos esquemas de segurança com XML. Basicamente, podemos usar dois tipos de criptografia: simétrica e assimétrica. A criptografia simétrica usa a mesma chave para converter o texto claro em texto criptografado e novamente em texto claro. A criptografia assimétrica usa uma chave para criptografar e uma outra chave para descriptografar. A criptografia assimétrica é mais segura porque uma das chaves nunca é transferida; no entanto, a criptografia simétrica é mais rápida.

O processo de implementar a segurança a nível da mensagem recorrendo à criptografia assimétrica começa com a criação de um par de chaves pública/privada. A chave pública é disponibilizada pelo fornecedor dos serviços para qualquer cliente que precise de a usar, mas mantém a chave privada em segredo. Estas duas chaves possuem uma relação matemática, mas é computacionalmente quase impossível calcular a chave privada a partir da chave pública. Uma mensagem encriptada com uma chave pública só pode ser desencriptada com a correspondente chave privada.

A figura 3.23 mostra como as chaves públicas e privadas actuam para encriptar uma mensagem.

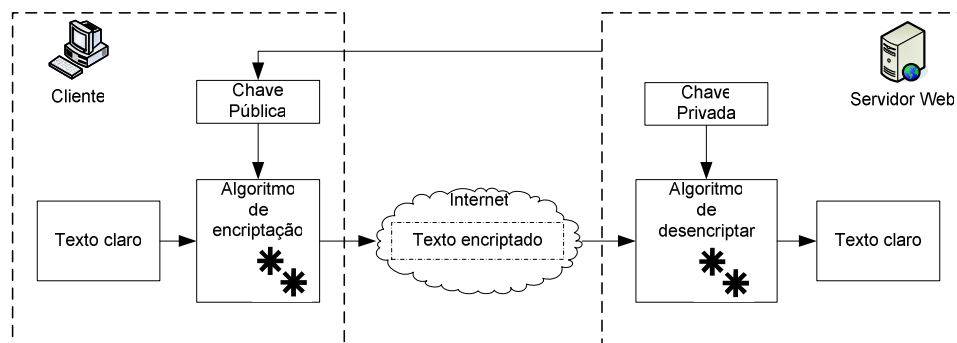


Figura 3.23 – Criptografia de chave pública/privada

Uma peça fundamental nesta infra-estrutura é o certificado de chave pública. Esses certificados são emitidos por empresas credíveis denominadas autoridades de certificação, que criam certificados em consonância com a norma *ITU-T X.509 Certificate Standard* [Santesson, 2003].

Para a abordagem da segurança ao nível da mensagem vamos subdividi-la em duas categorias: ao nível do XML e ao nível do SOAP.

Segurança ao nível do XML

A segurança ao nível do XML consiste em esquemas para encriptar e desencriptar documentos XML. O W3C publicou um conjunto de normas para segurança em XML, incluindo assinaturas digitais, encriptação e gestão de chaves.

Seguidamente apresentamos uma breve descrição dos principais mecanismos de segurança para XML.

XML-Signature

Fornece um esquema para representar assinaturas digitais num documento XML, garantindo desta forma quer a integridade dos dados quer a autenticação do cliente.

Este protocolo concretiza-se através de um conjunto de *tags* XML com informação que permite identificar o criador da transacção e averiguar se a mensagem foi modificada por entidades não autorizadas [W3C, 2005i].

XML-Signature permite que um determinado interveniente na transacção, ou nó, assine apenas uma parte do documento. Esta característica torna-se vital quando no decorrer do fluxo da transacção seja necessário o processamento de mensagens por vários nós.

XML-Encryption

XML-Encryption descreve a forma como encriptar documentos XML, especificando os métodos para encriptar as secções de um documento XML. As informações originais em texto claro são encriptadas e colocadas entre *tags* `<EncryptedData>` e, assim um documento XML que utiliza o *XML-Encryption* pode ser visto por qualquer entidade, mas apenas quem possuir a chave de descodificação conseguirá compreender o conteúdo que foi encriptado [W3C, 2005f].

XML Key Management Specification

Para que os mecanismos possam usar as chaves pública e privada alguns requisitos devem ser garantidos para que, por um lado, as aplicações possam validar as chaves e, por outro, possam comunicar automaticamente com as autoridades de certificação.

Para isso, o XKMS, fornece um protocolo baseado em XML/SOAP para distribuição, registo, verificação e renovação de chaves públicas [W3C, 2005h].

Security Assertion Markup Language

Um Web Service tem que ser capaz de avaliar se um cliente tem ou não autorização para o conectar e usar as suas operações da forma que pretende. Esta avaliação envolve os tópicos genericamente chamados de autenticação e autorização.

O SAML, permite que os clientes enviem as suas credenciais de autenticação e autorização aos Web Services. Com este protocolo os clientes podem transportar os seus direitos entre diferentes Web Services [OASIS, 2005b]. Esta característica é de extrema importância num ambiente com aplicações distribuídas que unifique vários Web Services.

eXtensible Access Control Markup Language

Como vimos, o SAML permite incluir informações nos documentos XML que comunicam as comprovações de autenticação e autorização. As decisões de autorização são baseadas em regras.

O protocolo XACML, define um vocabulário XML para declarar as regras para controlo de acesso num documento XML [OASIS, 2005a].

Segurança ao nível do SOAP

Nos pontos anteriores analisamos um conjunto de protocolos baseados em XML que fornecem métodos para garantir a confidencialidade e integridade dos dados, assim como, métodos para comunicar as credenciais de autenticação e autorização.

Para podermos usar estes protocolos em mensagens SOAP, um conjunto de especificações estão actualmente a serem desenvolvidas. Essas especificações publicadas com a denominação de *WS-Security* resultam de um consórcio de empresas e visam a construção de Web Services seguros promovendo a ampla interoperabilidade entre sistemas heterogéneos.

O *WS-Security* tem como principal objectivo integrar e unificar os vários modelos, mecanismos e tecnologias de segurança existentes, sendo dado desta forma mais um passo para a total interoperabilidade entre sistemas. As novas especificações de segurança definem um conjunto de normas que se concretizam em módulos/extensões que se podem associar ao cabeçalho de uma mensagem SOAP.

Na figura 3.24 podemos observar a pilha de especificações associadas do *WS-Security* [IBM e Microsoft, 2005] [Chappell, 2005].

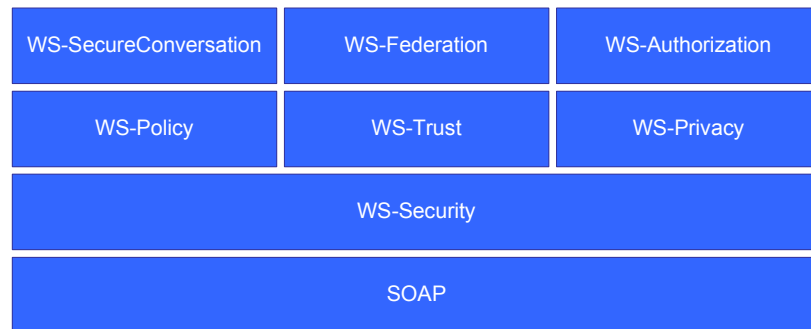


Figura 3.24 – Especificações associadas do WS-Security

O *WS-Security* descreve a forma como as assinaturas digitais e a encriptação podem ser colocadas nas mensagens SOAP, de forma a garantir a integridade e a confidencialidade das mesmas. Assim, os Web Services podem examinar as mensagens que recebem e, segundo as suas credenciais, aceitar ou rejeitar o processamento do pedido.

O *WS-Security* fornece a fundação para todas as outras especificações de segurança que tratam das políticas de segurança, relacionamentos de confiança, autenticação e autorização para criar um ambiente de interoperação seguro.

Apresentam-se de seguida, de forma bastante sucinta, os principais protocolos associados ao *WS-Security*.

WS-Policy – descreve como os agentes computacionais podem especificar as capacidades do Web Service, restrições de segurança e políticas de utilização do Web Service.

WS-Trust – descreve um modelo para estabelecer um relacionamento de confiança entre os intervenientes, quer directamente quer através de intermediários, para criar um ambiente seguro de interoperação.

WS-Privacy – descreve um modelo para definir como os Web Services serão adoptados e implementados, tanto para o fornecedor dos serviços como para o requisitante dos serviços.

WS-Secure Conversation – descreve como gerir e autenticar a troca de mensagens entre as partes intervenientes, incluindo especificações para um contexto de segurança e derivação de chaves de sessão.

WS-Federation – descreve como gerir os vários tipos de relacionamento em ambientes heterogéneos, incluindo o suporte das identidades associadas.

WS-Authorization – descreve como os Web Services farão a gestão dos dados e das políticas de autorização.

3.6.1.3 Comparação dos protocolos de segurança

Uma vez que todos os protocolos de segurança, apresentados na secção 3.6.1, podem ser usados em aplicações baseadas em Web Services, torna-se necessário conhecer as suas principais características, para que possamos escolher os que melhor se adaptam aos requisitos da arquitectura de cada projecto de Web Services.

Na tabela 3.2 podemos ver uma comparação dos protocolos de segurança abordados, para um conjunto de características de segurança.

Tabela 3.2 – Comparação dos protocolos de segurança

Característica	Protocolos					
	SSL	XML Signature	XML Encryption	XKMS	SAML	WS Security
Uso de XML	Não	Sim	Sim	Sim	Sim	Sim
Independência de plataforma	Não	Sim	Sim	Sim	Sim	Sim
Suporte a chaves PKI	Sim	Sim	Sim	Sim	Sim	Sim
Segurança extremo-a-extremo	Não	Sim	Sim	Sim	Sim	Sim
Integridade	Sim	Sim	Sim	Sim	Não	Sim
Autenticação da mensagem	Sim	Sim	Não	Sim	Não	Sim
Autenticação do Cliente	Sim	Sim	Não	Sim	Sim	Sim

3.6.2 Outros protocolos avançados

O universo dos protocolos avançados dos Web Services, para além dos protocolos para fornecer segurança a mensagens SOAP, deve também incluir protocolos para controlar o encaminhamento de mensagens e a gestão de transacções.

Na figura 3.25 podemos analisar a organização dos principais protocolos avançados dos Web Services. [MSDN, 2005]

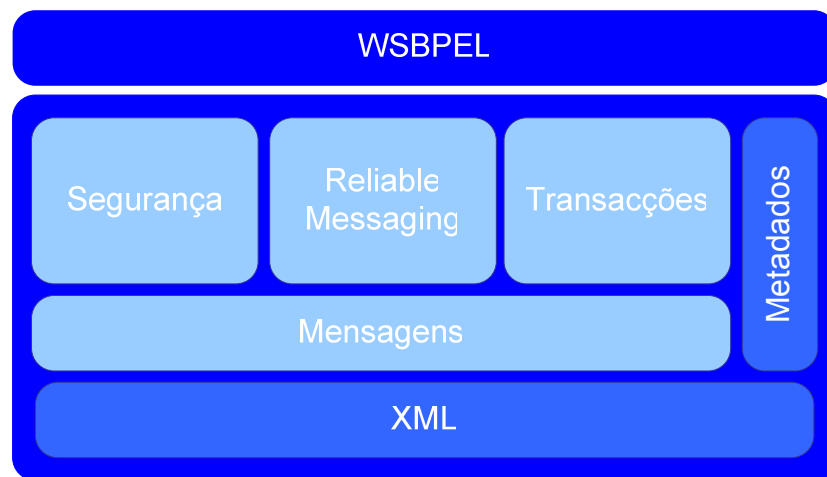


Figura 3.25 – Protocolos avançados dos Web Services

Seguidamente apresentamos uma breve descrição dos principais protocolos avançados dos Web Services que servem para assegurar o encaminhamento de mensagens, a gestão de transacções e garantir a qualidade de serviço.

WSBPEL – Corresponde a *Web Services Business Process Execution Language*, e especifica uma linguagem baseada em XML para definir processos comerciais.

WS-Transaction – define modelos e métodos baseados em XML para a gestão de transacções de Web Services, permitindo criar transacções atómicas e comerciais.

WS-Coordination – define uma gramática XML para coordenar requisições de Web Services, ou seja, como devem interagir para realizarem uma tarefa; por exemplo, o acordo no resultado de uma transacção em vários sistemas.

WS-Reliability – pretende introduzir a garantia de entrega nas mensagens SOAP; ao mesmo tempo que pretende assegurar a eliminação de mensagens duplicadas e dar garantia de que as mensagens chegam na ordem correcta.

3.7 Organizações que controlam as especificações

Neste capítulo, analisámos cada uma das especificações que formam o alicerce em que os Web Services estão fundados. A história da tecnologia mostra que se uma especificação estiver na posse apenas de uma empresa nem sempre o objectivo é melhorar a funcionalidade para o utilizador: muitas das vezes é tornar o utilizador refém e restringir-lhe a liberdade de usar produtos de outras empresas.

Após dezenas de guerras travadas nas duas últimas décadas, utilizadores e fornecedores de tecnologias buscam uma alternativa para a fúria da concorrência implacável. As principais empresas concordaram que um conjunto de especificações seriam de maior proveito para os utilizadores e também para elas.

Consequentemente, começaram a cooperar inseridas em consórcios sem fins lucrativos, cujo único objectivo é definir e publicar especificações. Seguidamente, apresentamos as organizações mais significativas no contexto geral da Internet e em particular dos Web Services.

3.7.1 W3C

O *World Wide Web Consortium*, W3C, é uma das mais importantes dessas organizações. No seu sitio Web, www.w3.org, o W3C afirma o seguinte:

O W3C desenvolve tecnologias interoperáveis, especificações, instruções, *software* e ferramentas para levar a Web ao seu potencial máximo. O W3C é um fórum de informações, comunicação e aprendizagem colectiva.

O W3C controla as especificações HTTP, XML, XML Schema, SOAP, WSDL, XML Encryption e XML Signature, entre outras. Superintende também o documento *WS-Architecture*, que pretende estabelecer uma definição formal dos Web Services.

3.7.2 OASIS

A *Organization for the Advancement of Structured Information Standards* é outra organização importante para os Web Services. No seu sitio Web, www.oasis-open.org, a OASIS afirma o seguinte:

A OASIS é um consórcio global sem fins lucrativos que promove o desenvolvimento, a convergência e a adopção de normas de comércio electrónico.

A OASIS controla as especificações UDDI, WS-Security, SAML e XACML, entre outras.

3.7.3 WS-I

A *Web Services Interoperability Organization* é uma organização recente, que surgiu após a definição do conceito dos Web Services. No seu sitio Web, www.ws-i.org, a WS-I define a sua missão:

A WS-I é uma organização de actividade aberta, licenciada para promover a interoperabilidade de Web Services entre plataformas, sistemas operativos e linguagens de programação. A organização trabalha com a indústria e com as organizações de normas para atender às necessidades dos clientes fornecendo orientação, melhores práticas e recursos para desenvolver soluções de Web Services.

A WS-I publica perfis, ferramentas de teste e aplicações de exemplo que garantem, na medida do possível, que diferentes normas funcionarão em conjunto. Um perfil é um conjunto das especificações que demonstraram ser capazes de funcionar bem em conjunto.

Na tabela 3.3 podemos analisar as principais especificações que compõem a arquitectura dos Web Services e, para cada especificação, qual a versão actual, em que data foi publicada a ultima revisão e a organização que a controla.

Tabela 3.3 – Especificações dos Web Services

Especificação	Versão	Data última revisão	Organização
HTTP	1.1	05/2000	W3C
XML	1.1	02/2004	W3C
SOAP	1.2	05/2003	W3C
WSDL	2.0	12/2004	W3C
WS-Architecture	1.0	02/2004	W3C
UDDI	3.0	02/2005	OASIS
SSL	3.0	11/1996	Netscape, IETF
XML-Encryption	1.0	12/2002	W3C
XML-Signature	1.0	02/2002	W3C
SAML	2.0	11/2004	OASIS
XACML	2.0	02/2005	OASIS
WS-Security	1.0	03/2004	OASIS
WS-Policy	1.0	08/2003	IBM, Microsoft, VeriSign, RSA, Etc..
WS-Trust	1.0	02/2005	IBM, Microsoft, CA, VeriSign, RSA, Etc..
WS-Privacy	1.0	08/2003	IBM, Microsoft, CA, VeriSign, RSA, Etc..
WS-SecureConversation	1.0	02/2005	IBM, Microsoft, CA, VeriSign, Etc..
WS-Federation	1.0	08/2003	IBM, Microsoft, VeriSign, RSA, Etc..
WS-Authorization	1.0	08/2003	IBM, Microsoft, CA, VeriSign, RSA, Etc..
WSBPPEL	1.1	05/2003	OASIS
WS-Transaction	1.0	11/2004	IBM, Microsoft, BEA
WS-Coordination	1.0	11/2004	IBM, Microsoft, BEA
WS-Reliability	1.1	11/2004	OASIS

3.9 Construção de Web Services

O desenvolvimento de aplicações baseadas na tecnologia dos Web Services seguindo o modelo da arquitectura orientada a serviços requer uma abordagem diferente da usada no desenvolvimento tradicional. Nesta secção apresentamos as principais etapas para o desenvolvimento de aplicações com Web Services.

3.9.1 Projecto de Web Services

Geralmente, grande parte do esforço do processo de desenvolvimento de Web Services é colocado na etapa de projecto. No projecto da aplicação definem-se os serviços, os tipos de dados e os formatos de mensagens, e como estes interagem.

O resultado da etapa de projecto deve ser representado num documento WSDL que descreve toda a interface. É através da informação contida nesse documento que as aplicações cliente são programadas para invocar as operações do Web Service.

3.9.2 Servidor

Os Web Services são um exemplo típico de aplicações orientadas pelo modelo cliente-servidor. É comum hoje em dia desenvolver aplicações usando uma abordagem de divisão do processo de desenvolvimento em três camadas: a camada de apresentação, a camada de aplicação e a camada de dados. A camada de apresentação é responsável pela interacção e apresentação dos dados ao utilizador. A camada de aplicação é responsável por implementar todas as funções de negócio. A camada de dados é responsável por implementar todas as operações de acesso aos dados.

No entanto, as operações de um Web Service, por definição, são implementadas para serem usadas programaticamente pelas aplicações cliente, portanto, não possuem interface gráfica, ou seja, não implementa a camada de apresentação.

Assim, um Web Service fica responsável por implementar apenas duas das camadas, a camada de aplicação e a camada de dados, i.e. as três camadas são distribuídas pelos dois sistemas envolvidos na comunicação, pelo servidor, com o Web Service, e pelo cliente, com a aplicação cliente.

Na figura 3.26 pode ser analisado um diagrama que demonstra a arquitectura de três camadas aplicada a um Web Service, considerando o modelo cliente-servidor.

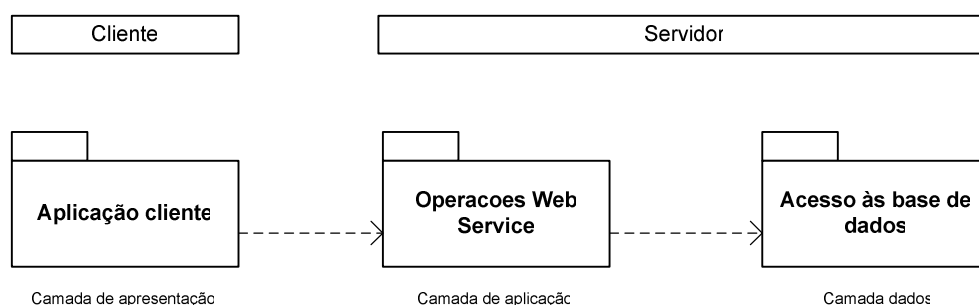


Figura 3.26 – Um Web Service visto como três camadas

Tendo o fornecedor dos serviços definido o projecto do Web Service e o modelo para desenvolvimento do Web Service, pode passar à etapa seguinte que consiste em codificar cada uma das operações disponibilizadas pelo Web Service, utilizando uma linguagem de programação.

De seguida, opcionalmente, pode publicar o Web Service registando-o nos repositórios do UDDI.

Apesar da forma de construção de Web Services ser naturalmente diferente em cada uma das plataformas existentes, podemos resumir as principais tarefas a executar, nos pontos seguintes:

- Criação de uma classe para implementar as funcionalidades do Web Service, ou seja, as operações públicas que serão expostas via *Web*, para acesso remoto;
- Definição dessa mesma classe como sendo do tipo Web Service.

3.9.3 Cliente

Depois do fornecedor dos serviços finalizar a construção do Web Service e proceder à sua publicação, qualquer outra organização é potencial requisitante desses serviços, podendo, para isso, usar uma ferramenta de *software* geral que permita invocar Web Services e trocar mensagens SOAP, ou então programar uma aplicação cliente personalizada para os seus requisitos e objectivos.

Na figura 3.27 podemos analisar as principais etapas que uma aplicação cliente desencadeia para descobrir, obter a descrição dos serviços e invocar as operações do Web Service, activando a interacção automática entre a aplicação cliente e o Web Service.

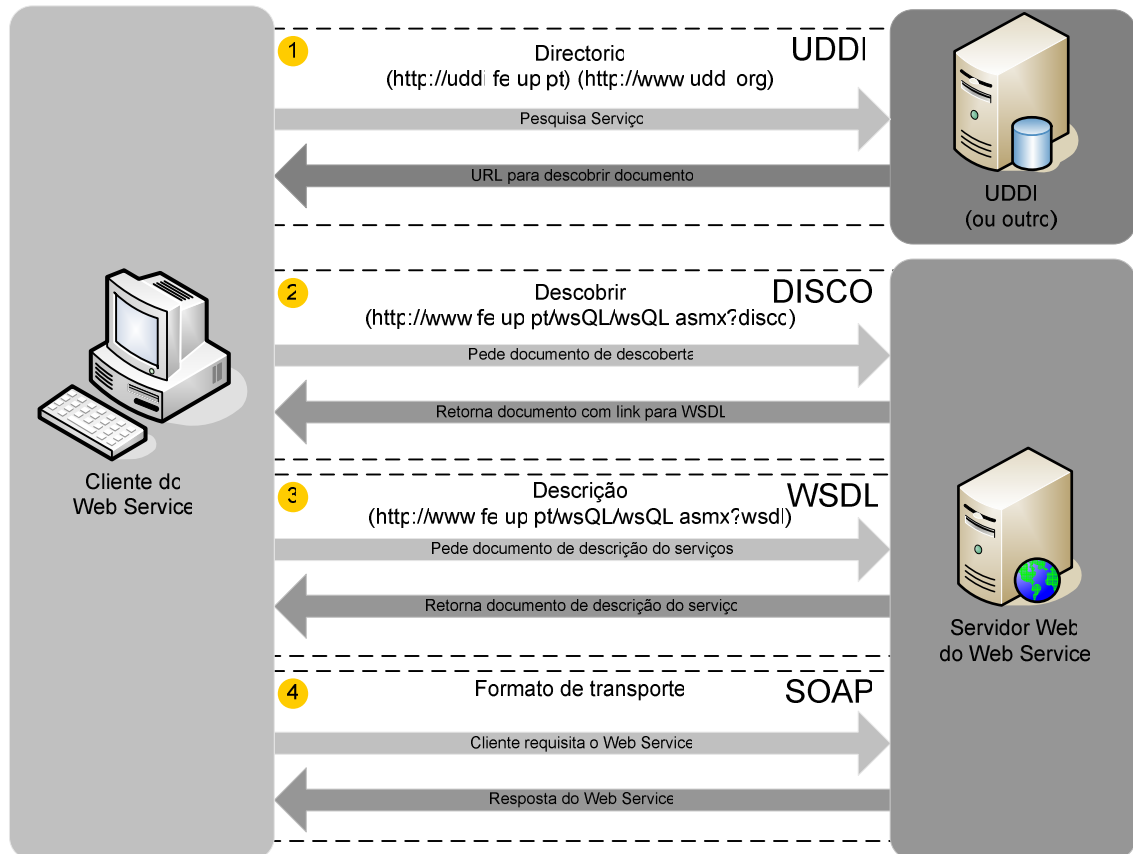


Figura 3.27 – Etapas de Descoberta, Descrição e Invocação de um Web Services

Façamos então uma descrição de cada uma das etapas efectuadas por uma aplicação cliente para programar a interacção com um Web Service:

1. Procura o Web Service no UDDI ou então obtém o URI junto do fornecedor do Web Service;
2. Utilizando o URI obtém o documento de descoberta que permite requisitar o documento WSDL;
3. Obtém o documento WSDL que contém toda a descrição dos serviços;
4. Cria um *proxy* local do Web Service e, depois, invoca as operações do Web Service através de mensagens SOAP.

Especificação da Aplicação

4.1 Apresentação da aplicação wsQL

O período da história em que vivemos ficará caracterizado como uma época de globalização crescente, onde as Tecnologias de Informação e Comunicação, e em especial a Internet, emergem para assumir uma posição privilegiada para a comunicação entre aplicações, com particular ênfase na integração de tecnologias e sistemas computacionais. Mas, apesar da infra-estrutura da Internet se encontrar implantada a nível mundial e uma década após o seu aparecimento, das transformações sofridas, para aumentar o seu escopo e potencial, podemos concluir que, actualmente, a maior parte das actividades efectuadas no ambiente da Internet são feitas numa lógica de interacção homem – aplicações.

Mas em muitos casos, o cenário ideal era um ambiente onde fosse possível a interacção automática entre aplicações, utilizando a infra-estrutura mundial que a Internet oferece. Desta forma, concretizando em pleno as potencialidades anunciadas por Tim Berners-Lee, um dos criadores do conceito da Web, que o definiu como: “um espaço onde a informação tem um significado bem definido, facilitando a cooperação entre pessoas e agentes computacionais”.

A tecnologia dos Web Services aparece como sendo capaz de activar esta interacção automática entre agentes computacionais. É neste contexto que pretendemos desenvolver a aplicação de suporte a esta tese.

A ideia do desenvolvimento desta aplicação surgiu inicialmente com o objectivo principal de explorar e demonstrar as potencialidades da tecnologia emergente, os Web Services, na integração de aplicações, e para isso seria utilizado um problema simulado, com uma organização tipo com um determinado sistema de informação.

No entanto, após a realização do estudo inicial, reconhecemos que podíamos ser mais ambiciosos e desenvolver uma aplicação de âmbito geral, passível de ser utilizada em qualquer organização e para qualquer sistema de informação, num contexto distribuído, utilizando a teia mundial da Internet para suportar a comunicação entre as aplicações.

Com isto, pretendemos aproveitar o facto de, actualmente, praticamente qualquer organização possuir um sistema de informação mais ou menos complexo e possuir uma rede informática interna e ligação à Internet, sendo essa ligação cada vez mais rápida.

Por outro lado, a história dos sistemas de informação mostra que as aplicações que constituem um sistema de informação são, na maior parte das vezes, incapazes de trocarem dados entre si, criando assim ilhas de dados, quer internamente, quer entre empresas, a uma escala global. Tomámos consciência de que seria uma mais valia, para qualquer organização, poder abrir o seu sistema de informação aos sistemas de informação dos seus parceiros de negócio. Desta forma, uma organização pode oferecer a terceiros um pseudo controlo do seu sistema de informação.

Portanto, para levar a cabo esta missão, foi necessário fazer algumas escolhas: entre as principais, decidir desenvolver uma aplicação para uma organização em particular ou desenvolver uma aplicação que possa ser usada por qualquer organização, e decidir implementar a integração de sistemas de informação baseados num determinado sistema de gestão de base de dados, o que nos colocava perante a tomada de outra decisão, que era escolher qual o sistema de gestão de base de dados, ou então implementar a integração de sistemas de informação baseados em qualquer sistema de gestão de base de dados.

As opções tomadas em resposta a estas questões anteriores foram no sentido de tornar a aplicação o mais flexível possível, e assim, a aplicação a desenvolver deveria ser suficientemente geral para ser implementada em qualquer organização, com sistemas de informação baseados em qualquer sistema de gestão de base de dados.

4.2 Objectivos da aplicação wsQL

A aplicação a desenvolver insere-se na categoria de sistemas de informação distribuídos, aproveitando a infra-estrutura que a Internet disponibiliza e será construída segundo o modelo de arquitectura orientada a serviços, através da tecnologia de Web Services.

O objectivo principal desta aplicação é permitir a uma organização expor um conjunto de funções, que permitam receber comandos para interrogação ou manutenção de uma base de dados, suportando, assim, a interacção automática entre aplicações, num ambiente distribuído utilizando a Internet como espinha dorsal para a comunicação.

Este capítulo descreve a actividade da análise de requisitos dessa aplicação cujo nome é wsQL – Web Service Query Language. Na figura 4.1 podemos observar um diagrama que representa uma visão geral da aplicação acima definida.

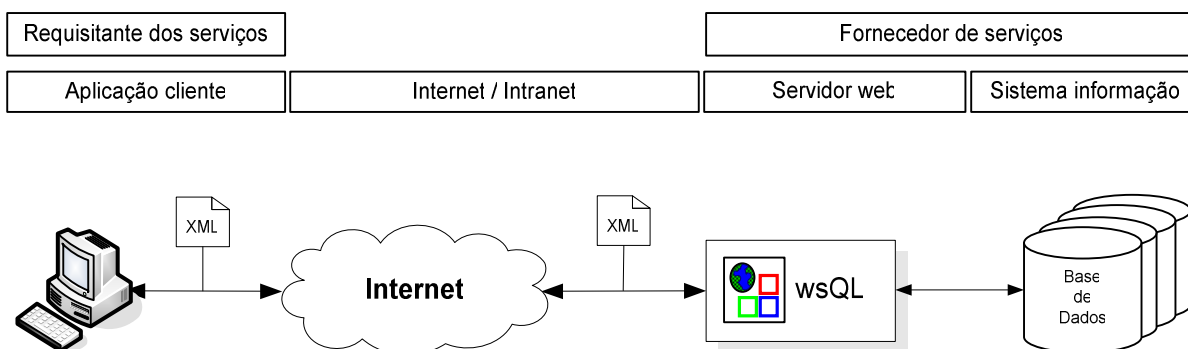


Figura 4.1 – Visão geral da aplicação wsQL

A aplicação wsQL pode ser considerada como uma nova camada de *software*, que uma organização coloca entre o seu sistema de informação e o sistema de informação dos seus parceiros de negócio, quer sejam internos ou externos, permitindo, assim, a total integração de aplicações entre a organização fornecedora dos serviços, ou funções, e a das dos seus parceiros, requisitantes do serviços. Esse cenário está ilustrado na figura 4.2.

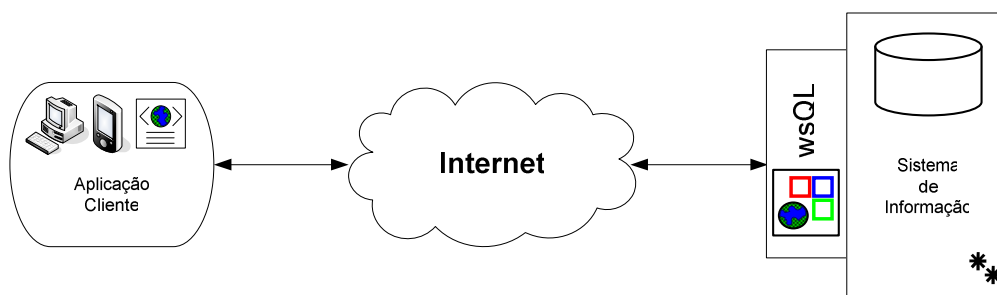


Figura 4.2 – A aplicação wsQL vista como uma nova camada de *software*

A aplicação wsQL será implementada com um conjunto de operações, ou funções, que possibilite a interacção automática entre as aplicações intervenientes na integração, respectivamente a aplicação cliente e a aplicação fornecedora do serviço, ou seja, o Web Service. Essas funções serão disponibilizadas como serviços, que poderão ser utilizados por terceiros, com a devida permissão, através de aplicações programadas a partir da informação contida no documento WSDL, para usar essas funções remotas como se de funções locais se tratasse. Na figura 4.3 podemos observar esta situação.

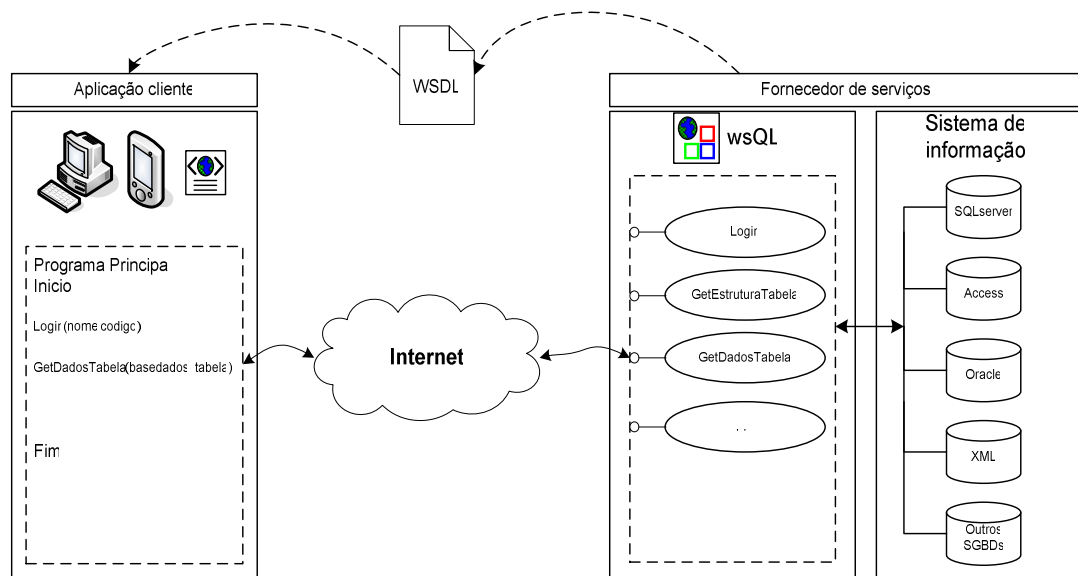


Figura 4.3 – Programação da interacção da aplicação cliente com a aplicação wsQL

Para apresentar as principais funcionalidades desta aplicação vamos agrupá-las em duas categorias: funcionalidades para o gestor da aplicação e funcionalidades para a aplicação cliente que usa as funções da aplicação wsQL.

Funcionalidades para o gestor da aplicação

- Manutenção de utilizadores da aplicação;
- Partilha de base de dados e tabelas pelos utilizadores;
- Definição de permissões de selecção, inserção, actualização e eliminação de registos dos utilizadores para cada tabela partilhada;
- Consultar os utilizadores que estão a usar a aplicação a um dado momento;
- Consultar o histórico de ligações e operações efectuadas por cada utilizador.

Funcionalidades para a aplicação cliente

- Requisitar à aplicação wsQL a lista de bases de dados e tabelas e respectivas permissões, a que pode aceder;
- Requisitar à aplicação wsQL os registos de uma determinada tabela;
- Requisitar à aplicação wsQL a estrutura de uma determinada tabela;
- Efectuar a manutenção de registos numa determinada tabela, desde que tenha permissão para isso;
- Submeter comandos SQL a uma base de dados;
- Consultar o histórico de ligações e operações efectuadas e sucesso das mesmas.

4.3 Requisitos da aplicação

Com a aplicação wsQL, pretende-se dotar uma determinada organização com um sistema computacional que permita disponibilizar acesso às suas base de dados internas, a terceiros, devidamente autorizados, para uma simples consulta ou então para um procedimento de manutenção de registos, de uma forma segura.

O acesso à base de dados é efectuado através de um conjunto de funções, empacotadas na aplicação wsQL, que será instalada na organização. Essas funções serão utilizadas por outras organizações, através de aplicações programadas para invocar essas funções.

Portanto, uma organização, ao utilizar a aplicação wsQL, fica com o total controlo sobre o sistema, uma vez que todas as funções vitais de negócio estão na organização, podendo, assim, decidir quem tem permissão para usar as funções da aplicação wsQL, e a que base de dados tem acesso cada aplicação cliente, e para cada base de dados a que tabelas pode aceder, definindo também o que pode fazer em cada tabela, ou seja, se pode inserir, alterar e ou eliminar registos.

Seguidamente, apresentamos os principais requisitos da aplicação wsQL, agrupados em requisitos funcionais e requisitos não funcionais.

4.3.1 Requisitos funcionais

A aplicação wsQL deve disponibilizar um conjunto de funções para implementar o acesso a base de dados de vários sistemas de gestão de base de dados, por parte de outras aplicações, permitindo seleccionar, inserir, actualizar e eliminar registos. Para isso, deve definir uma interface normalizada que encapsule toda a complexidade da programação e que forneça a interacção automática entre a aplicação cliente e o Web Service wsQL.

A aplicação wsQL deve receber a invocação, por parte da aplicação cliente, de uma função com os respectivos parâmetros, de seguida tem de transformar essa invocação no conjunto de comandos necessários para submeter o pedido ao sistema de gestão de base de dados e, depois, tem de devolver a resposta à aplicação cliente.

Assim, a aplicação não possui uma interface gráfica, visto que o objectivo é fornecer um conjunto de funções para serem utilizadas programaticamente através de aplicações cliente.

4.3.2 Requisitos não funcionais

Apesar da aplicação wsQL não possuir, por definição do conceito de Web Service, uma interface gráfica, terá que ser construída uma ferramenta de *software* que permita fazer a gestão da aplicação. Essa ferramenta deve ser uma aplicação com interface *Web* para permitir também a gestão remota. Os principais procedimentos de gestão da aplicação são: criação de utilizadores e a gestão das partilhas de base de dados; no entanto outros procedimentos de consulta e controlo devem também estar presentes.

Uma vez que, teoricamente, qualquer aplicação pode ser programada para utilizar as funções da aplicação wsQL, será necessário prever uma forma de controlar o acesso à aplicação, por exemplo através de um código de acesso. Por outro lado, como a aplicação será usada num contexto aberto da *Web*, e a comunicação entre as aplicações será implementada através de mensagens SOAP seguindo as regras do XML, é necessário garantir a confidencialidade dos dados. Basicamente, para assegurar a segurança dos dados podemos usar esquemas de encriptação que pode ser efectuada ao nível do protocolo de transporte ou ao nível da mensagem.

Num outro nível de segurança, a aplicação wsQL deve garantir que a aplicação cliente apenas acede às bases de dados e às tabelas que o gestor do sistema autorizou, e

permitir apenas as operações definidas, ou seja, se pode seleccionar, inserir, alterar e eliminar registos.

4.3.3 Casos de utilização

Para documentar as diversas fases do processo de desenvolvimento da aplicação, vamos utilizar uma abordagem de elaboração de sistemas de informação orientados por objectos. Assim, para representar os requisitos da aplicação wsQL, em especial os requisitos funcionais, recorreremos a diagramas de casos de uso da notação UML.

Com a aplicação wsQL podem interagir dois tipos de utilizadores ou actores, o gestor do sistema e aplicação cliente, e fazem-no de forma bastante distinta: o primeiro tem uma função de gestão e controlo da aplicação, a segunda, depois de obtida a autorização do gestor do sistema, pode interagir com a aplicação utilizando as funções que esta disponibiliza. De seguida fazemos uma breve descrição de cada um destes actores:

- **Gestor do sistema** – é o responsável na organização onde está instalada a aplicação wsQL por, através da interface *Web* construída, fazer a gestão dos utilizadores, ou seja, definir, para cada utilizador, a que bases de dados tem acesso, para cada base de dados a que tabelas pode aceder, e para cada tabela o que pode fazer: seleccionar, inserir, actualizar ou apagar registos.
- **Aplicação cliente** – aplicação de *software* com permissão para usar, remota ou localmente, as funções definidas na aplicação wsQL, com as restrições definidas pelo gestor.

Tomando como referência cada um dos actores, nos pontos seguintes identificamos e descrevemos os casos de utilização em que participam.

4.3.3.1 Casos de utilização: gestor do sistema

O gestor do sistema é um utilizador humano devidamente credenciado para efectuar a gestão e controlo da aplicação wsQL. Como tal, a sua interacção com o sistema é efectuada numa lógica de utilizador - aplicação. Basicamente tem acesso à parte passiva do sistema constituído pela aplicação wsQL.

Nos pontos seguintes estão definidos os casos de utilização para o gestor do sistema:

- Aceder à aplicação wsQL com perfil de gestor;
- Gestão de utilizadores;
- Partilhar base de dados / tabelas;
- Consultar ligações *online*;
- Consultar histórico ligações de utilizadores;
- Consultar histórico operações de utilizadores;
- Terminar sessão.

Na figura 4.4 estão representados todos os casos de utilização nos quais o gestor do sistema pode participar.

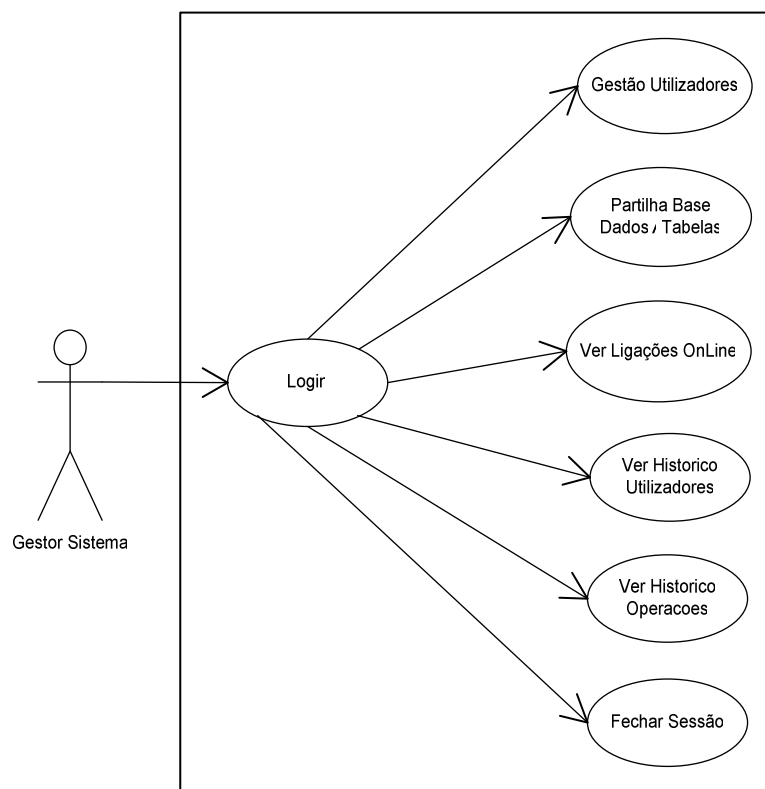


Figura 4.4 – Diagrama de casos de utilização: gestor do sistema

Descrição dos casos de utilização do gestor do sistema:

- Aceder ao wsQL com perfil de gestor
 1. o gestor do sistema, ou seja, o responsável pela aplicação wsQL na organização, pode aceder através de uma interface *Web* para configuração da aplicação, para isso, deve informar um nome de utilizador e um código de acesso;
 2. devem ser mostradas todas as opções de configuração da aplicação wsQL.
- Gestão de utilizadores
 1. o gestor do sistema pode inserir, alterar ou eliminar dados pessoais relativos aos utilizadores da aplicação wsQL, dados que incluem nome da empresa, o nome do responsável, o nome de utilizador e o código de acesso.
- Partilhar base de dados / tabelas
 1. o gestor do sistema deve para cada utilizador definir a que bases de dados tem acesso, para cada base de dados a que tabelas pode aceder, e para cada tabela o que pode fazer: seleccionar, inserir, actualizar ou apagar registos.
- Consultar ligações *online*
 1. o gestor do sistema pode consultar os utilizadores, ou seja saber quais as aplicações cliente que estão a usar o wsQL num determinado momento e que operações estão a efectuar.
- Consultar o histórico das ligações de utilizadores
 1. o gestor do sistema pode consultar para cada utilizador o histórico das utilizações do wsQL.
- Consultar o histórico das operações efectuadas
 1. o gestor do sistema pode consultar para cada utilizador o histórico das operações efectuadas em cada uma das base de dados a que este pode aceder.
- Terminar sessão
 1. o gestor de sistema deve fechar a sessão quando finalizar as tarefas de gestão e controlo.

4.3.3.2 Casos de utilização: aplicação cliente

A aplicação cliente é um utilizador computacional devidamente autorizado para manipular as bases de dados da organização, através da utilização das funções da aplicação wsQL. Como tal, a sua interacção com o sistema é efectuada numa lógica de aplicação – aplicação de forma automática. Basicamente a aplicação cliente tem acesso à parte activa do sistema constituído pela aplicação wsQL.

Nos pontos seguintes estão definidos os casos de utilização para a aplicação cliente:

- Aceder à aplicação wsQL e transferir os parâmetros;
- Transferir estrutura de uma tabela;
- Transferir registos de uma tabela;
- Inserir / alterar / apagar registos de uma tabela;
- Executar comandos SQL na base de dados;
- Consultar o histórico de ligações;
- Consultar o histórico de operações efectuadas;
- Terminar ligação.

Na figura 4.5 estão representados todos os casos de utilização em que a aplicação cliente pode participar.

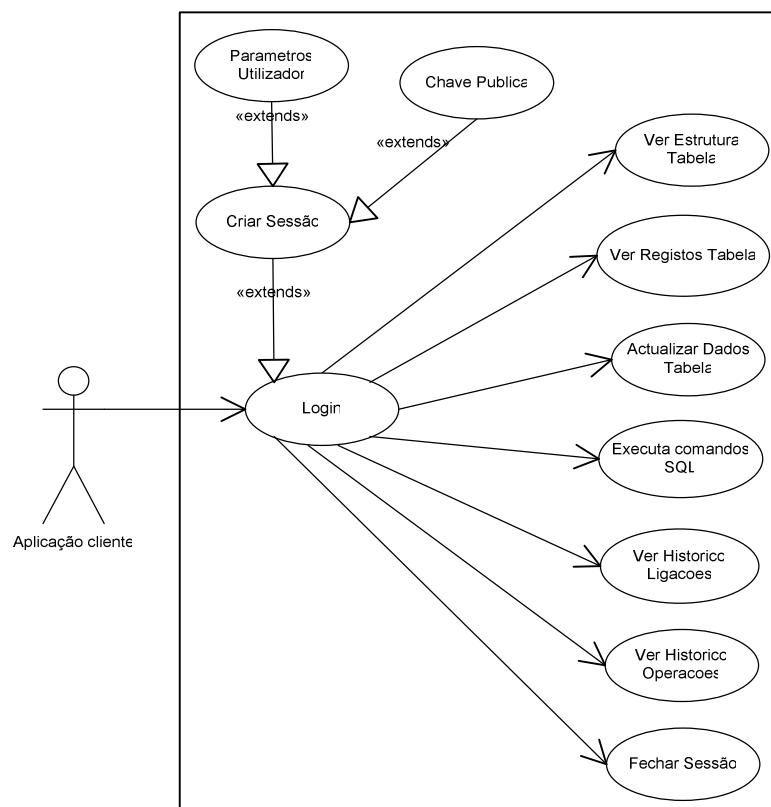


Figura 4.5 – Diagrama de casos de utilização: aplicação cliente

Descrição dos casos de utilização da aplicação cliente:

- Aceder à aplicação wsQL e transferir os parâmetros
 1. a aplicação cliente deve ligar-se ao wsQL e requisitar a chave pública para encriptar os dados a enviar;
 2. de seguida deve invocar a função *Login* enviando o nome do utilizador e código de acesso encriptado com a chave pública;
 3. caso o utilizador seja validado, um ficheiro XML será retornado para a aplicação cliente, com as seguintes informações:
 - IDutilizador – um identificador temporário para a presente ligação;
 - Nome – nome do utilizador registado;
 - Empresa – nome da empresa, caso se aplique;
 - UltimoAcesso – data do último acesso ao wsQL;
 - IDbd – identificador de cada base dados a que o utilizador tem acesso;
 - BaseDados – nome de cada base dados a que o utilizador tem acesso;
 - Tabela – nome de cada tabela para cada base de dados a que o utilizador tem acesso;
 - Seleccionar – para cada tabela informa se pode seleccionar registos;
 - Inserir – para cada tabela informa se pode inserir registos;
 - Actualizar – para cada tabela informa se pode alterar registos;
 - Apagar – para cada tabela informa se pode apagar registos.
 4. com estas informações a aplicação cliente fica com todos os parâmetros necessários para efectuar operações de consulta e manutenção nas tabelas definidas.
- Transferir estrutura de uma tabela
 1. a aplicação cliente deve invocar a função *GetEstruturaTabela* do wsQL e indicar qual a tabela;
 2. a estrutura da tabela deve ser retornada à aplicação cliente em formato XML.
- Transferir registos de uma tabela
 1. a aplicação cliente deve invocar a função *GetDadosTabela* do wsQL e indicar qual a tabela;

2. todos os registos da tabela devem ser devolvidos para a aplicação cliente em formato XML.
- Inserir / alterar / apagar registos de uma tabela
 1. a aplicação cliente deve invocar a função *ActualizaDadosTabela* do wsQL e indicar qual a tabela e quais as alterações;
 2. os registos da tabela devem ser alterados de acordo com o pedido da aplicação cliente.
 - Executar comandos SQL na base de dados
 1. a aplicação cliente deve invocar a função *ExecutaSQL* do wsQL e indicar quais os comandos SQL a executar;
 2. os comandos SQL enviados devem ser validados a nível de sintaxe e a nível de permissões para efectuar as operações pedidas nas tabelas;
 3. caso sejam validos, os comandos SQL devem ser executados de acordo com o pedido da aplicação cliente;
 4. caso se aplique, o resultado deve ser devolvido à aplicação cliente.
 - Consultar o histórico de ligações
 1. a aplicação cliente deve invocar a função *HistoricoLigacoes* do wsQL;
 2. o histórico de ligações deve ser retornado em formato XML, com as seguintes informações para cada ligação:
Início – data e hora de ligação ao wsQL;
Fim – data e hora de finalização da ligação ao wsQL.
 - Consultar o histórico de operações efectuadas
 1. a aplicação cliente deve invocar a função *HistoricoOperacoes* do wsQL;
 2. o histórico de operações efectuadas pela aplicação cliente deve ser retornado em formato XML, com as seguintes informações para cada operação:
Data – data e hora da invocação da função;
Operação – nome da função usada;
BaseDados – nome da base de dados;
Tabela – este campo pode conter o nome da tabela à qual a operação foi submetida ou o comando SQL pedido na operação;
Sucesso – informa se a operação teve ou não sucesso.

- Terminar ligação
 1. a aplicação cliente deve invocar a função *FecharSessao* do wsQL;
 2. a ligação deve ser fechada e se for caso disso, retornada a indicação de sucesso.

4.4 Funcionamento lógico da aplicação wsQL

Esta aplicação será construída para ser usada num contexto distribuído utilizando os protocolos da Internet e programada segundo a arquitectura dos Web Services. Basicamente, funcionará de forma análoga à requisição de uma página *Web*, no entanto, de modo diferente de uma página *Web*, os dados serão estruturados em XML e a aplicação cliente não será o *browser*, mas sim, uma aplicação *Web* ou *desktop*, programada para aceder ao Web Service.

Portanto, estamos perante um típico sistema distribuído, cuja arquitectura se baseia no paradigma do modelo de Cliente – Servidor; em que o Cliente, representa a aplicação que invoca o serviço, e o Servidor representa a aplicação que presta o serviço.

O funcionamento de qualquer Web Service, portanto, também da aplicação wsQL, requer três etapas essenciais: Publicação, Descoberta e Invocação. Claro que, uma outra etapa indispensável terá que ocorrer antes de qualquer das anteriores: a programação das operações que serão disponibilizadas pelo Web Service. Nessa etapa reside a nossa missão.

4.4.1 Servidor

A aplicação wsQL é constituída por todas as operações que o fornecedor de serviços disponibiliza; assim, a aplicação para poder ser usada de qualquer parte do mundo através da Internet deve ser instalada num servidor *Web* como qualquer outra aplicação *Web* tradicional. Desta forma, todas as operações disponibilizadas pela aplicação serão invocadas pelas aplicações cliente através de pedidos HTTP utilizando o URI da aplicação wsQL.

Das etapas indicadas anteriormente, o fornecedor dos serviços é responsável pela publicação do Web Service no UDDI; no entanto, a publicação é opcional, e caso não seja feita, a aplicação não pode ser encontrada automaticamente por outras aplicações e, neste

caso, cada potencial utilizador do Web Service deve contactar directamente o gestor da aplicação a fim de obter a forma da sua aplicação se ligar, ou seja, requisitar o documento WSDL.

4.4.2 Cliente

A aplicação que pretenda utilizar as operações disponibilizadas pela aplicação wsQL procura a mesma no directório do UDDI, caso tenha sido publicada, ou então requisita o documento WSDL directamente através do URI do fornecedor dos serviços que aponta para esse documento, como mostrado no seguinte exemplo:

<http://www.fe.up.pt/wsQL/wsQL.asmx?wsdl>

Desta forma a aplicação cliente tem acesso a toda a interface da aplicação wsQL, obtendo a descrição de todas as operações com os respectivos parâmetros e tipos de mensagens que permitem aceder a essas operações. Assim, a aplicação cliente pode invocar remotamente as operações do wsQL, activando a interacção automática entre as duas aplicações.

Das etapas indicadas anteriormente, a aplicação cliente é responsável pela descoberta da aplicação, requisição da descrição das operações da aplicação e invocação das operações que deseja utilizar.

4.4.3 Diagramas de actividades

Para representar a dinâmica do sistema recorreremos a diagramas de actividades da notação UML. Para cada entidade interveniente neste sistema computacional, isto é, para o gestor da aplicação e a aplicação cliente, apresentamos inicialmente um diagrama geral que representa todo o fluxo de actividades envolvendo todos os casos de utilização da entidade. Para os casos de uso que exijam mais detalhes devido à sua complexidade será apresentado um diagrama que descreve o fluxo de actividades.

4.4.3.1 Gestor do sistema

No diagrama de actividades da figura 4.6 estão representadas as acções de gestão e controlo da aplicação wsQL que serão executadas pelo responsável do sistema na organização. Para levar a cabo essas acções a aplicação wsQL terá que ser dotada de um sistema de informação que permita fazer a gestão dos utilizadores, definir as bases de dados e tabelas disponíveis para cada utilizador, com as respectivas permissões de selecção, inserção, alteração e eliminação de registos.

O diagrama pretende mostrar todas as acções disponíveis para o gestor do sistema; portanto, a sequência das acções pode ser qualquer após ser efectuada a validação do gestor.

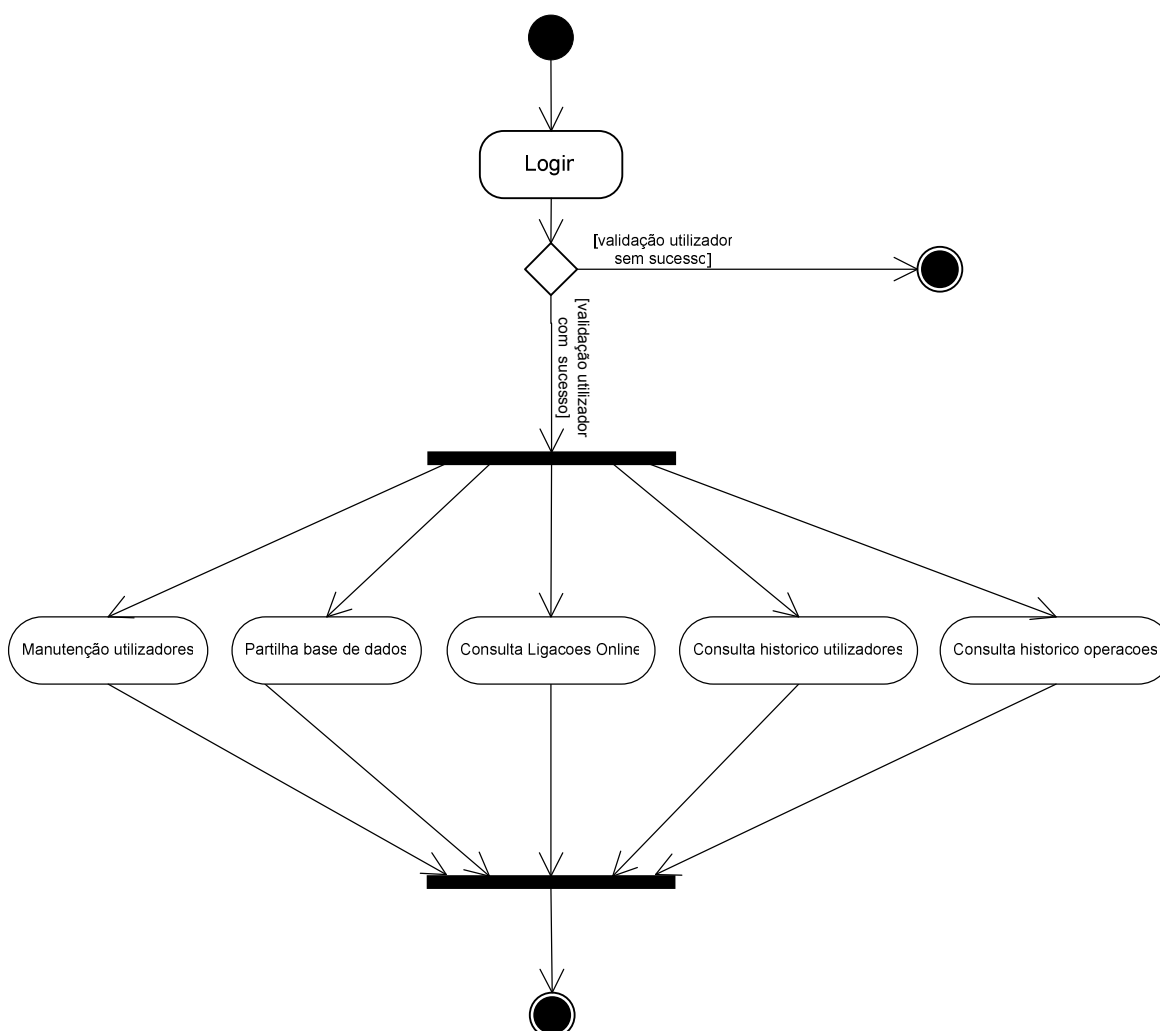


Figura 4.6 – Diagrama de actividades: casos de uso do gestor sistema

4.4.3.2 Aplicação cliente

No diagrama de actividades da figura 4.7 estão representados todos os casos de usos que a aplicação cliente pode realizar depois de efectuada a validação perante a aplicação wsQL e de ter recebido a lista de parâmetros que descreve as permissões que o gestor do sistema da organização fornecedora definiu para essa aplicação cliente.

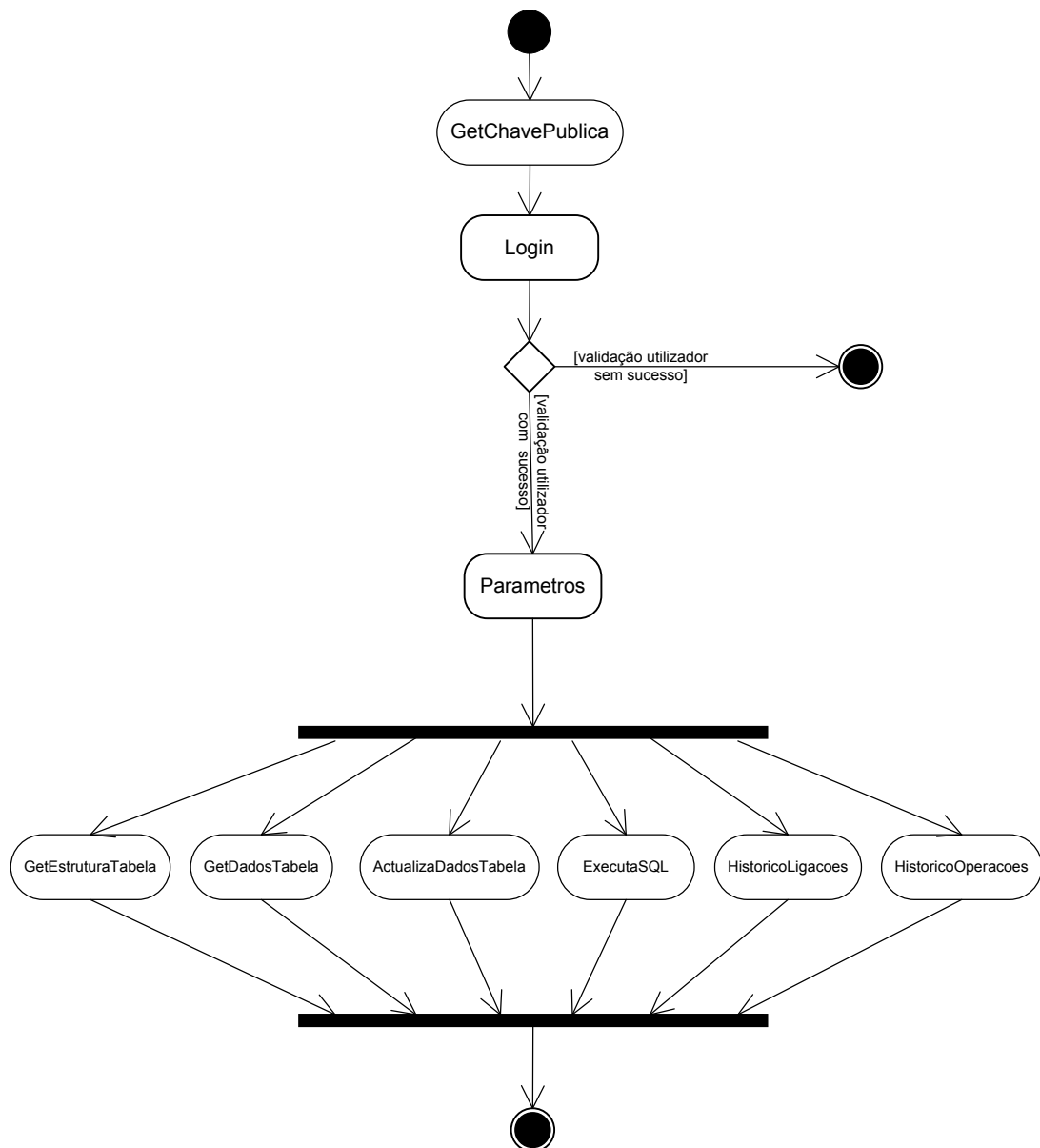


Figura 4.7 – Diagrama de actividades: casos de uso da aplicação cliente

Na figura 4.8 temos um diagrama de actividades que decompõe em sub-actividades as actividades *GetEstruturaTabela*, *GetDadosTabela*, *ActualizaDadosTabela*.

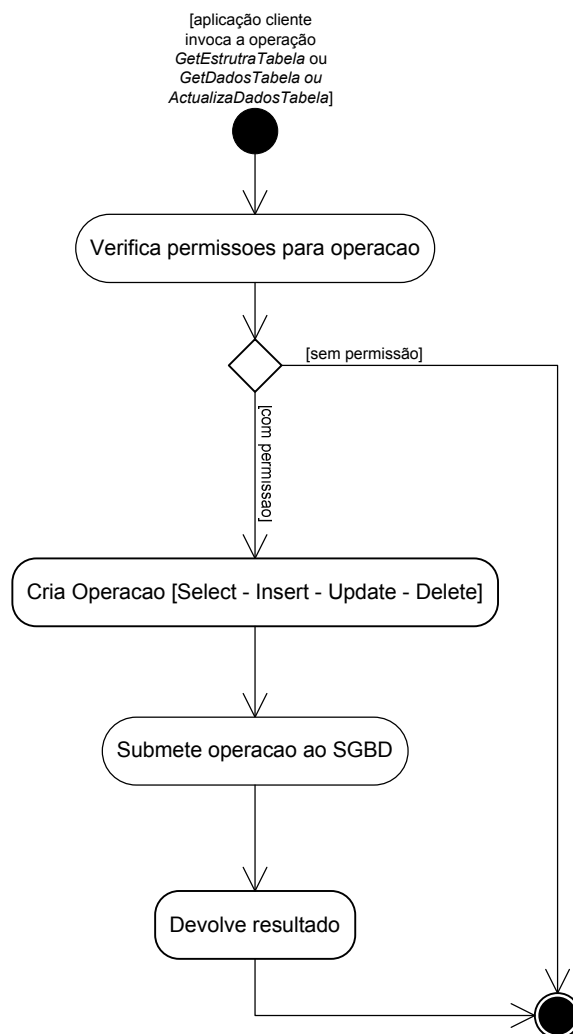


Figura 4.8 – Diagrama de actividades: casos de utilização
GetEstruturaTabela, *GetDadosTabela*, *ActualizaDadosTabela*

Este diagrama descreve as actividades que são accionadas quando a aplicação cliente pretende seleccionar, inserir, alterar ou eliminar registos de uma determinada tabela. Basicamente, é verificado se a aplicação cliente que está a invocar a operação relativamente a uma tabela, tem permissão atribuída pelo gestor do sistema para efectuar essa operação.

O diagrama anterior mostra as principais acções a executar para um conjunto de operações predefinidas de selecção e actualização de registos; no entanto, a aplicação cliente pode submeter instruções SQL através da operação *ExecutaSQL*. Devido à liberdade oferecida à aplicação cliente de submeter instruções SQL, outras acções são necessárias para verificação e correcção, se possível, da sintaxe da instrução SQL.

A figura 4.9 ilustra o conjunto de actividades a executar quando a aplicação cliente submete instruções SQL para a aplicação wsQL interpretar e submeter ao sistema de gestão de base de dados.

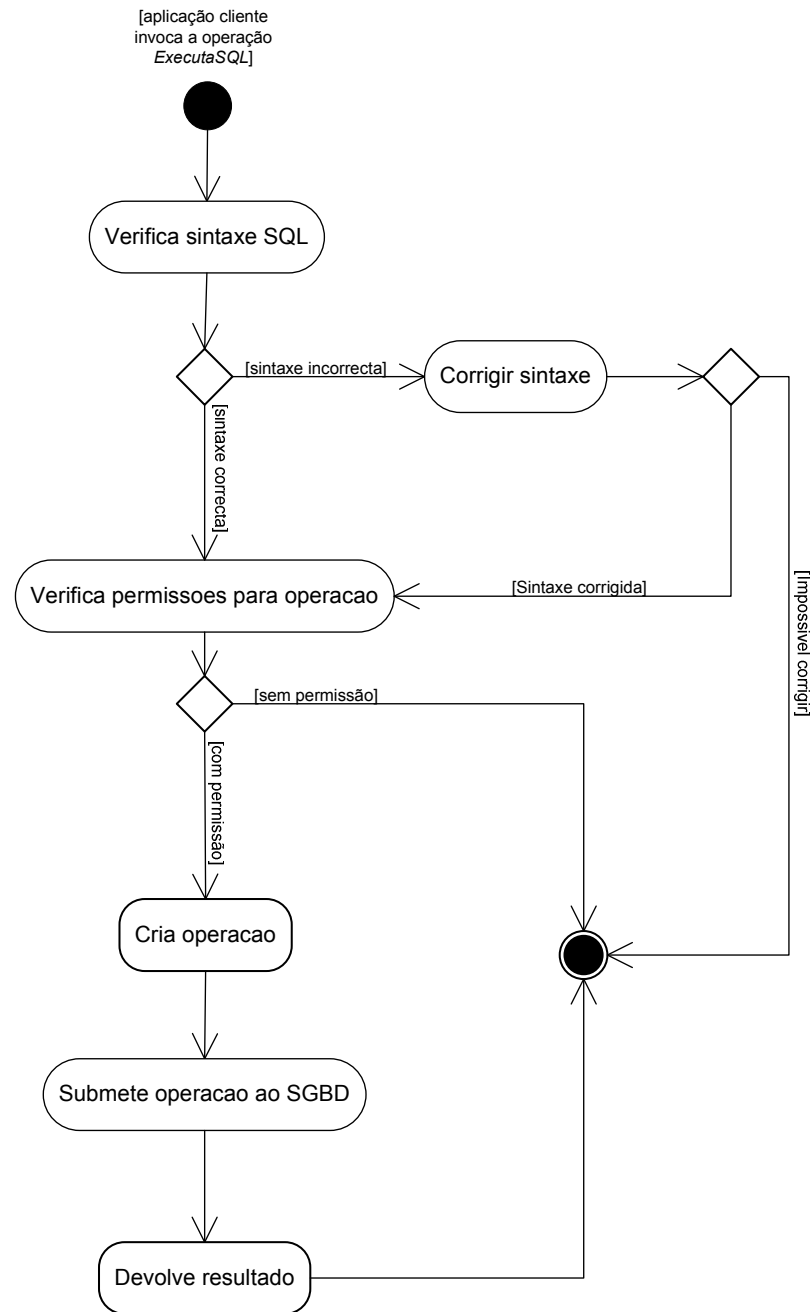


Figura 4.9 – Diagrama de actividades: caso de utilização *ExecutaSQL*

4.4.4 Diagramas de sequência

Para representar o comportamento e a interacção entre os objectos do sistema recorremos ao diagrama de sequência da notação UML.

4.4.4.1 Gestor do sistema

Na figura 4.10 apresentamos um diagrama de sequência que representa todos os casos de uso que o gestor do sistema pode efectuar.

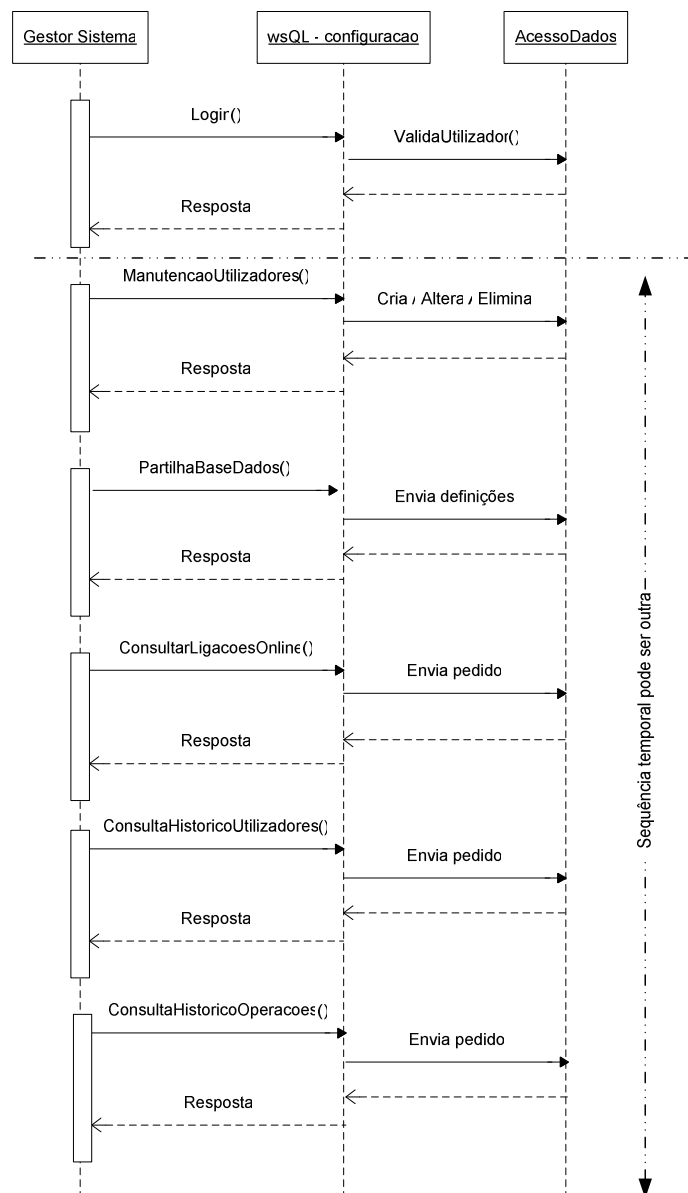


Figura 4.10 – Diagrama de sequência: casos de uso do gestor do sistema

4.4.4.2 Aplicação cliente

Na figura 4.11 apresentamos um diagrama de sequência que representa todos os casos de uso que a aplicação cliente pode efectuar.

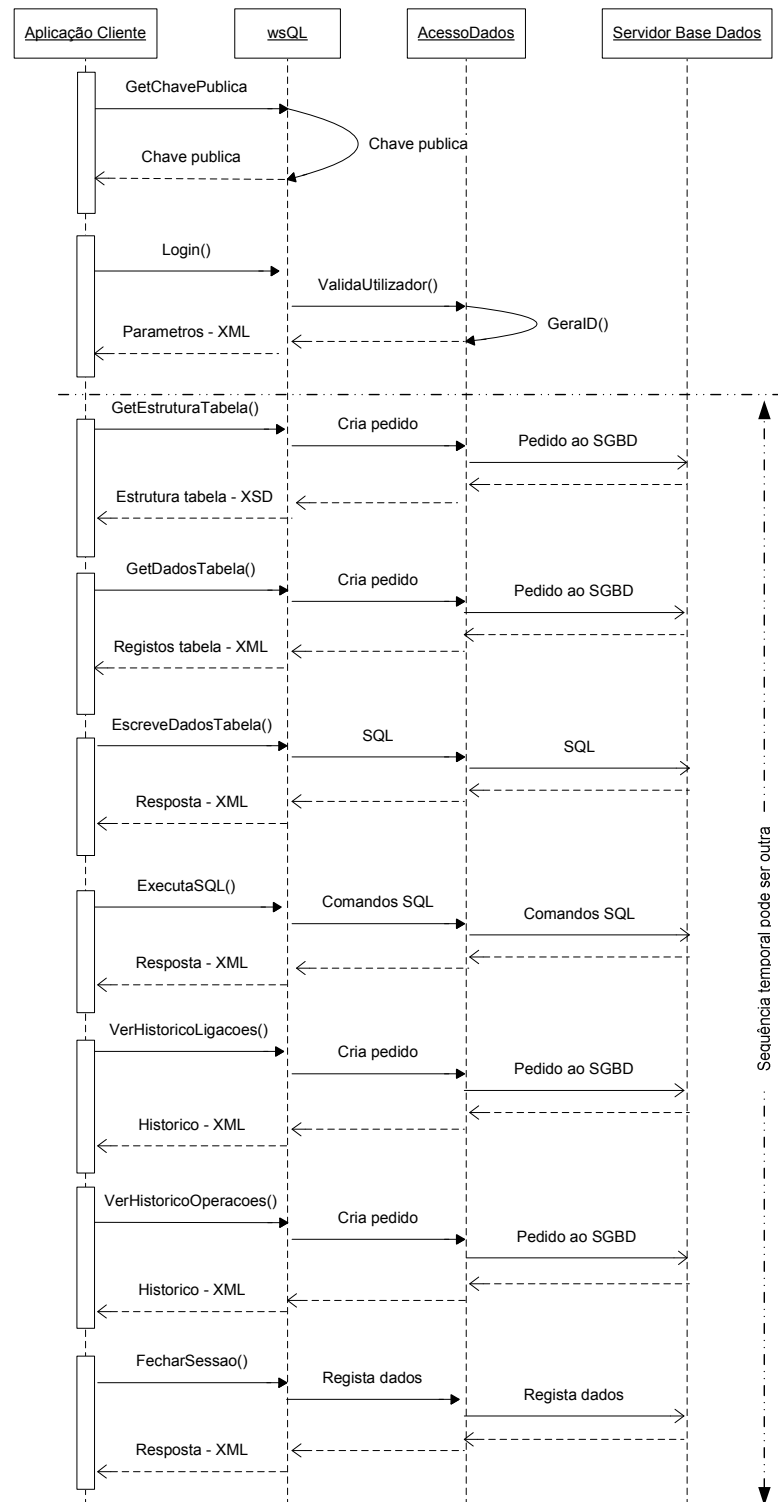


Figura 4.11 – Diagrama de sequência: casos de uso da aplicação cliente

4.4.5 Etapas para a utilização

Qualquer organização que pretenda abrir o seu sistema de informação, em particular as suas bases de dados, a parceiros de negócio pode utilizar a aplicação wsQL. Desta forma, uma aplicação de um parceiro pode manipular as bases de dados da organização através das operações disponibilizadas pela aplicação wsQL.

A aplicação wsQL é constituída pelo Web Service propriamente dito, que tem por objectivo expor as operações via Internet, e por um módulo de gestão e controlo, que tem por objectivo permitir ao gestor da aplicação efectuar a manutenção de utilizadores, definir quais as bases de dados e tabelas a disponibilizar para cada utilizador e as permissões deste sobre cada tabela.

Neste ponto indicamos as etapas principais a efectuar pelo fornecedor dos serviços para disponibilizar o acesso às suas bases de dados e as etapas para a aplicação cliente aceder a essas bases de dados.

As principais etapas para uma organização fornecer estes serviços através da aplicação wsQL podem ser subdivididas em etapas de instalação e em etapas de gestão e controlo.

Instalação

1. Instalar a aplicação wsQL no servidor *Web*;
2. Opcionalmente, publicar a aplicação no registo do UDDI.

Gestão e controlo

1. Definir quais as bases de dados a disponibilizar;
2. Manutenção de utilizadores;
3. Definir para cada utilizador a que base de dados e tabelas pode aceder e o que pode fazer: seleccionar, inserir, alterar e eliminar registos;
4. Consultas/controlo das ligações e operações efectuadas por parte das aplicações cliente.

Na figura 4.12 estão representadas as principais etapas para uma organização usar a aplicação wsQL.

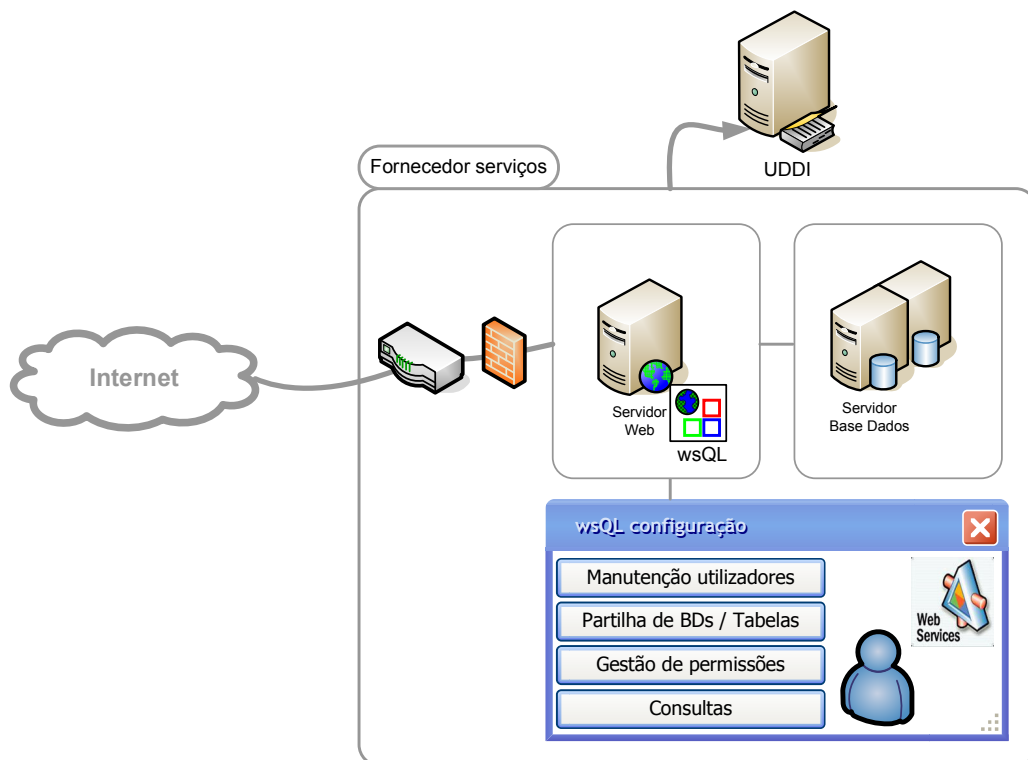


Figura 4.12 – Aplicação wsQL no fornecedor dos serviços

Ultrapassadas as etapas anteriores, qualquer organização pode programar aplicações para aceder à aplicação wsQL e utilizar os seus serviços, isto é, as suas operações, podendo assim manipular as bases de dados da organização fornecedora dos serviços, de acordo com as regras definidas pelo gestor do sistema.

Assim, as principais etapas para que uma aplicação cliente possa usar as funcionalidades do Web Service wsQL, ou seja, possa aceder às base de dados da organização que instala a aplicação wsQL, podem ser agrupadas em etapas de programação da aplicação e etapas de utilização das operações da aplicação wsQL.

Programação da aplicação

1. Procurar a aplicação wsQL no UDDI ou então obter o URI junto do fornecedor da aplicação wsQL;
2. Utilizando o URI obter o documento WSDL (anexo D);
3. Escolher a plataforma para a construção da aplicação cliente;
4. Criar uma instância da aplicação wsQL, (existem plataformas que executam este processo automaticamente a partir do documento WSDL);
5. Fornecer os dados ao fornecedor do wsQL para que este indique um nome de utilizador e um código de acesso.

Utilização das operações da aplicação wsQL

1. Sempre que a aplicação cliente se liga ao wsQL, a primeira operação a invocar é *getChavePublica* que devolve a chave pública para encriptar os dados a enviar.
2. Seguidamente deve invocar a operação *Login* e fornecer como parâmetros o nome do utilizador e código de acesso encriptado. Caso obtenha sucesso na validação recebe um documento XML contendo um identificador temporário para a ligação e toda a informação sobre as bases de dados e tabelas com as respectivas permissões a que a aplicação cliente pode aceder.

Na figura 4.13 estão representadas as principais etapas para que uma aplicação cliente possa aceder à aplicação wsQL e utilizar as funcionalidades disponibilizadas.

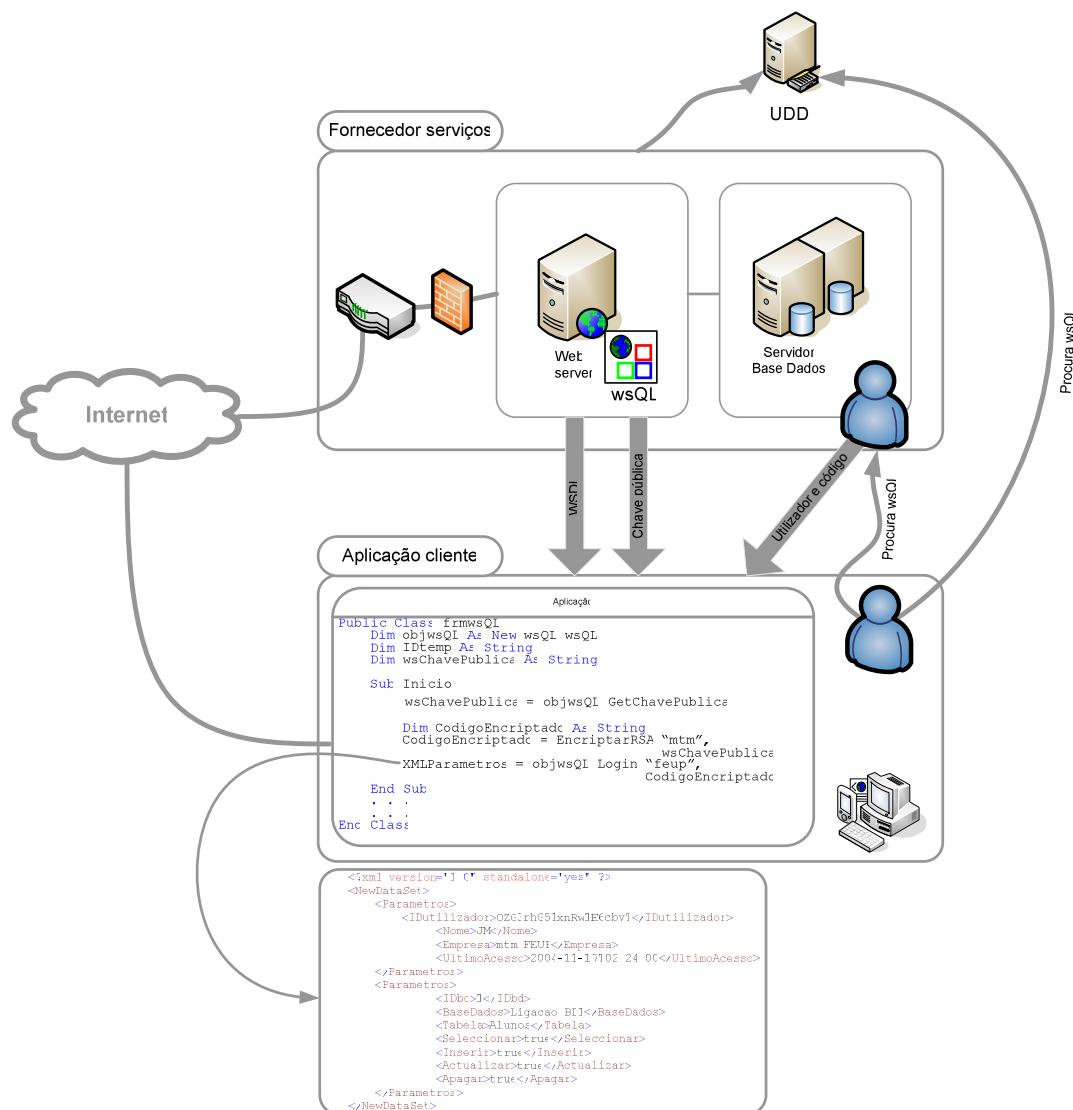


Figura 4.13 – Programação da aplicação cliente

Desenvolvimento da Aplicação

5.1 Introdução

Como foi analisado no capítulo anterior, o principal objectivo do sistema wsQL – Web Service Query Language é tornar possível que duas ou mais aplicações possam *falar* entre si, actuando assim, como um tradutor entre ambas, activando, desta forma, a interacção automática entre aplicações num cenário distribuído, e tendo como espinha dorsal para a comunicação a Internet.

O presente capítulo pretende descrever todo o processo de desenvolvimento do sistema wsQL que sustenta os conceitos apresentados na dissertação.

O processo de desenvolvimento deste sistema de informação foi orientado pelos modelos iterativo e incremental, essencialmente para comportar a inclusão de novas funcionalidades, numa lógica de melhoria contínua.

Em termos de actividade de desenvolvimento do sistema wsQL, este será abordado subdividindo-o em três subsistemas principais:

- **Subsistema do Web Service** – este subsistema é responsável por albergar as operações que são expostas via Internet e que são utilizadas remotamente por outras aplicações. É através destas operações que a aplicação cliente acede às bases de dados da organização, isto é, estas operações actuam como sendo a interface entre a aplicação cliente e as funções de negócio da organização fornecedora dos serviços.
- **Subsistema de acesso aos sistemas de gestão de base de dados** – neste subsistema são implementadas todas as funções para acesso e manutenção de vários sistemas de gestão de base de dados. Este subsistema permite, por um lado, que a organização fornecedora dos serviços possa disponibilizar acesso a

base de dados de vários sistemas e, por outro lado, a aplicação cliente não tem de saber com que sistema de gestão de base de dados está a lidar, simplesmente usa a operação.

- **Subsistema de gestão da aplicação** – a função principal deste subsistema é criar uma interface gráfica com ferramentas de gestão e controlo disponíveis para o gestor do sistema na organização fornecedora dos serviços. As tarefas de gestão incluem manutenção de utilizadores, definir quais as bases de dados e tabelas disponíveis a cada utilizador e definição das permissões de selecção, inserção, alteração e eliminação de registos, para cada tabela disponível a cada utilizador. Tarefas de controlo incluem monitorar as ligações e respectivas operações efectuadas pelas aplicações cliente.

Qualquer um destes subsistemas foi desenvolvido com base nos requisitos levantados no capítulo quatro, quer nos funcionais, quer nos não funcionais, e nos casos de uso identificados para cada um dos utilizadores ou actores que podem interagir com o sistema: a aplicação cliente e o gestor do sistema.

O três subsistemas identificados anteriormente e as relações entre eles podem ser observados na figura 5.1.

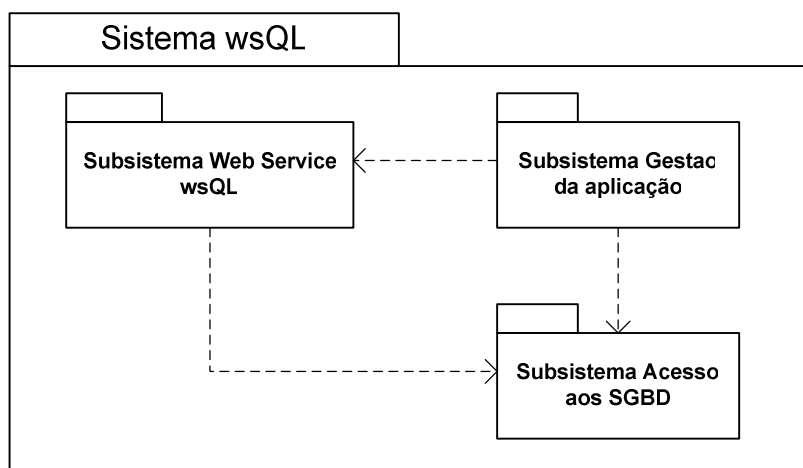


Figura 5.1 – Subsistemas do sistema wsQL

5.1.1 Método de desenvolvimento

O rumo imprimido no capítulo quarto, com a análise orientada por objectos, é consolidado neste capítulo consagrado ao desenvolvimento do sistema wsQL.

Como está patente na figura 5.1, a actividade de codificação deste sistema foi baseada na programação de componentes autónomos, para cumprir dois objectivos: o primeiro, era ser possível a reutilização noutros sistemas, por exemplo, o componente de acesso aos dados será usado pelas operações do Web Service e pela aplicação de gestão da aplicação; o segundo, era permitir uma melhoria contínua, por exemplo, no caso do Web Service, incluir novas operações, no caso do componente de acesso aos dados, incluir acesso a novos sistemas de gestão de bases de dados.

Podemos resumir as principais etapas de codificação de todo o sistema wsQL, nas seguintes:

- **Definição e codificação do Web Service** – consiste basicamente numa primeira fase definir quais as operações que estarão disponíveis a aplicações cliente através de protocolos da Internet. Faz parte desta definição caracterizar aspectos como determinar quais os parâmetros de entrada e quais os dados de saída que serão retornados à aplicação cliente. Numa segunda fase programar as funções que activam as funcionalidades disponíveis nas operações do Web Service. Este tipo de codificação terá que seguir as regras da arquitectura dos Web Services.
- **Codificação do acesso aos SGBDs** – consiste em programar todas as funções necessárias para aceder e efectuar a manutenção de tabelas dos sistemas de gestão de base de dados.
- **Codificação da aplicação de gestão do sistema** – consiste em criar uma aplicação com uma interface gráfica que permita ao gestor do sistema da organização onde o Web Service é implantado efectuar a gestão e o controlo do mesmo, de acordo com os requisitos identificados.

Este sistema é para ser usado num contexto distribuído, basicamente seguindo o modelo cliente – servidor. Contudo, toda a descrição anterior refere-se apenas à construção da parte do servidor. Portanto, fez também parte da nossa missão a elaboração de uma aplicação cliente que fosse capaz de demonstrar todas as capacidades do Web Service wsQL.

5.1.2 Plataforma de implementação

Para a implementação do sistema, a escolha recaiu sobre a plataforma .NET da Microsoft, pelo facto de, por um lado, a plataforma .NET ter sido projectada desde o início para incluir suporte aos Web Services, tanto na vertente de criação como na vertente de consumo de Web Services. Os Web Services constituem uma parte significativa da plataforma .NET. A Microsoft está a apostar o futuro da empresa na plataforma .NET e no sucesso dos Web Services. Por outro lado, temos o facto do autor do sistema se encontrar bastante à vontade com esta plataforma, porque, entre outras razões, fez alguns estudos sobre esta plataforma durante o percurso lectivo deste curso de mestrado.

Para utilização em pleno da plataforma .NET, a Microsoft criou um ambiente de desenvolvimento integrado, o Visual Studio .NET, que contém um conjunto de ferramentas para a criação de todo o tipo de aplicações, com especial ênfase nas aplicações baseadas em Web Services.

A plataforma .NET representa uma mudança drástica nos métodos de desenvolvimento de *software* tradicionais da Microsoft. Toda a organização da estrutura da plataforma .NET é semelhante à plataforma Java, sendo as principais diferenças sentidas ao nível das camadas inferior e superior [Hanson, 2003].

A plataforma Java é multiplataforma ao nível de arquitectura de hardware e do sistema operativo, enquanto que a plataforma .NET é dependente do sistema operativo Windows, embora existam actualmente alguns projectos para a implementar noutras plataformas tecnológicas.

A nível de linguagens de programação a plataforma Java é dependente da linguagem Java, enquanto que a plataforma .NET é multilinguagem, suportando entre muitas mais, VB.NET, C# e mesmo o J# com sintaxe Java, uma aplicação pode mesmo ser programada utilizando várias linguagens. A estrutura da plataforma .NET foi projectada para aceitar novas linguagens e novos compiladores.

Portanto, a codificação do sistema foi realizada beneficiando das potencialidades da plataforma .NET, utilizando como linguagem de programação VB .NET.

5.2 Subsistema do Web Service

Este subsistema constitui o núcleo principal de todo o sistema a que chamamos wsQL – Web Service Query Language.

A construção do Web Service foi orientada por uma abordagem de programação de componentes autónomos, adoptando o paradigma de desenvolvimento de aplicações seguindo o modelo de três camadas, que são a camada de apresentação, a camada de aplicação e a camada de dados.

Contudo, como um Web Service, por definição, não possui interface gráfica com o utilizador, isto porque as suas operações são definidas para serem usadas programaticamente pelas aplicações cliente, um Web Service, não implementa as funções da camada de apresentação.

Portanto, um Web Service implementa apenas duas das camadas, a camada de aplicação e a camada de dados, o que significa que as três camadas serão distribuídas pelos dois intervenientes na comunicação, o servidor, com o Web Service, e pelo cliente, com a aplicação cliente.

Assim sendo, neste Web Service, na camada de aplicação, serão definidas as operações que a aplicação cliente pode usar, funcionando desta forma como interface de ligação entre a aplicação cliente e o Web Service. Também são definidas todas as funções que implementam as regras fundamentais da aplicação, ou seja, esta camada recebe os pedidos da aplicação cliente através das operações definidas e, através das regras projectadas, invoca as operações da camada de dados.

Na camada de dados são definidas todas as funções que implementam o acesso a vários sistemas de gestão de base de dados, para selecção e manutenção de dados.

Na figura 5.2 pode ser visto um diagrama que ilustra a arquitectura de três camadas usada no sistema wsQL, considerando o modelo cliente-servidor.

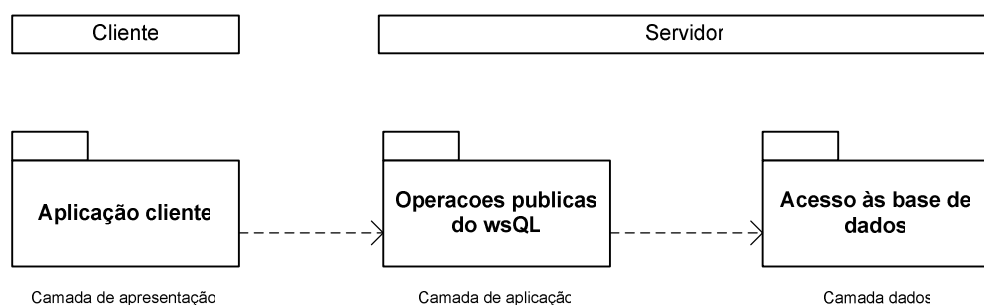


Figura 5.2 – O sistema wsQL visto em três camadas

Neste típico sistema cliente-servidor implementado através da tecnologia dos Web Services, as funções das três camadas foram distribuídas por dois computadores. Nesse sentido, o computador servidor trata das funções das camadas de aplicação e de dados, e o computador cliente, com a aplicação cliente, trata das funções da camada de apresentação que envolvem a comunicação e interação com o utilizador.

5.2.1 Definição da arquitectura do sistema wsQL

Neste ponto definimos a arquitectura do sistema wsQL para satisfazer os requisitos identificados. Nesse sentido, esta fase consiste em determinar quais as operações necessárias em cada uma das camadas.

Assim, na camada de aplicação são definidas as operações públicas a expor via protocolos da Internet, operações estas que serão encapsulados no componente wsQL. Num componente a que chamamos *Modulo* serão programadas todas as funções que na prática implementam as funcionalidades das operações públicas definidas.

A camada de dados é responsável pela implementação das funções que realmente permitem que as operações do componente wsQL comuniquem com bases de dados de vários sistemas de gestão de bases de dados. Essas funções são programadas no componente a que chamamos *AcessoBaseDados*.

Este isolamento das funções em camadas separadas visa promover a reutilização, escalabilidade e manutenção dos componentes de forma independente.

Na tabela 5.1 podemos analisar a organização lógica das operações definidas para cada camada.

Tabela 5.1 – Operações definidas para cada uma das camadas

wsQL	AcessoBaseDados
GetChavePublica	TipoSGBD
Login	AbreBaseDados
GetEstruturaTabela	CriaDataSet
GetDadosTabela	ExecutaINSERT
ActualizaDadosTabela	ExecutaUPDATE
ExecutaSQL	ExecutaDELETE
HistoricoLigacoes	UpdateDataSet
HistoricoOperacoes	
Camada de aplicação	Camada de dados

Nesta tabela omitimos propositadamente um conjunto de funções implementadas no componente *Modulo*, que desempenham um papel auxiliar na implementação das regras da camada de aplicação.

Na figura 5.3 podemos analisar um diagrama completo que demonstra a arquitectura que definimos para o sistema wsQL.

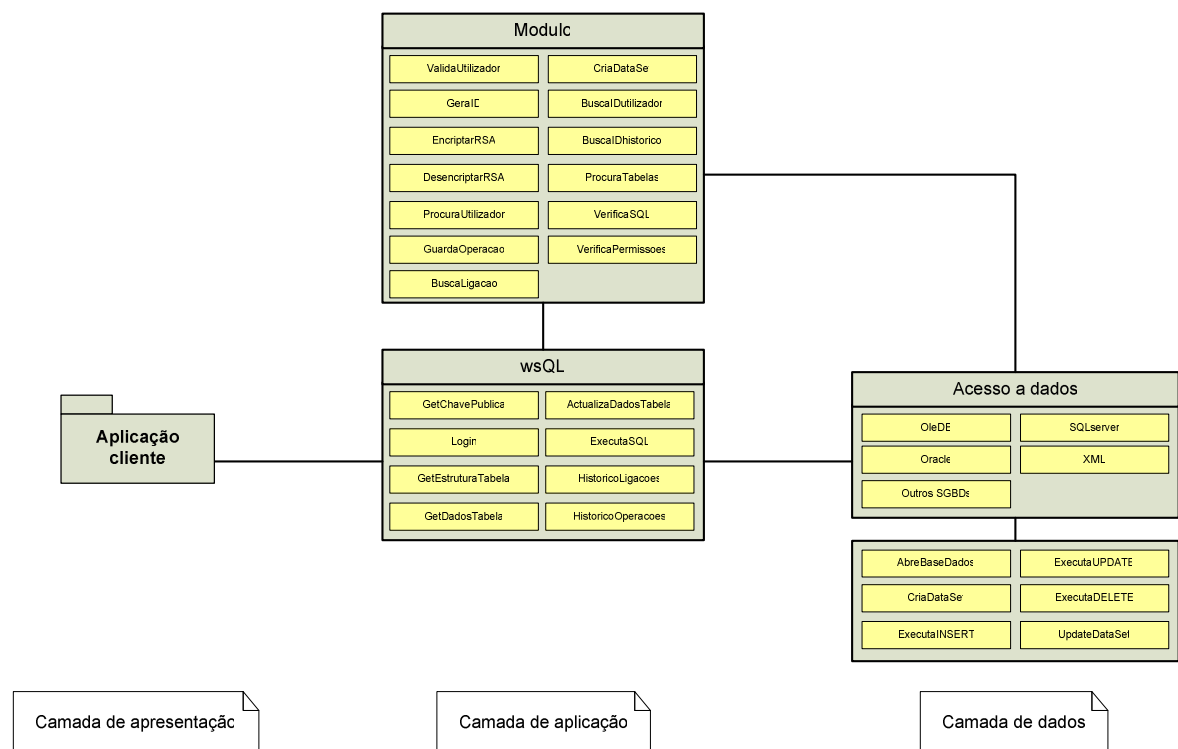


Figura 5.3 – Arquitectura do sistema wsQL

As operações definidas anteriormente são concretizadas através de um conjunto de componentes programados autonomamente. Na figura 5.4 temos o diagrama de componentes e as suas relações, para o modelo definido, considerando apenas as duas camadas do Web Service.



Figura 5.4 – Diagrama de componentes do Web Service wsQL

Até aqui apresentámos a arquitectura do sistema, com a definição das operações públicas, e os principais componentes a implementar; de seguida vamos entrar na fase de construção propriamente dita do Web Service, baseada no modelo de classes. Nesse diagrama podemos ver as principais classes com as respectivas operações e para cada operação, os dados de entrada e os dados de saída.

5.2.2 Modelo de classes

Através do diagrama de classes UML pretendemos representar os requisitos funcionais do sistema, que foram levantados na fase de análise. A partir deste diagrama será feita a codificação do sistema.

As classes identificadas para o Web Service wsQL foram:

- Classe wsQL;
- Classe Modulo.

Cada uma destas classes será caracterizada pelo seu conjunto de operações associando-lhes os respectivos parâmetros.

Neste primeiro diagrama de classes que pode ser analisado na figura 5.5, apresentamos apenas as classes da camada de aplicação do sistema.

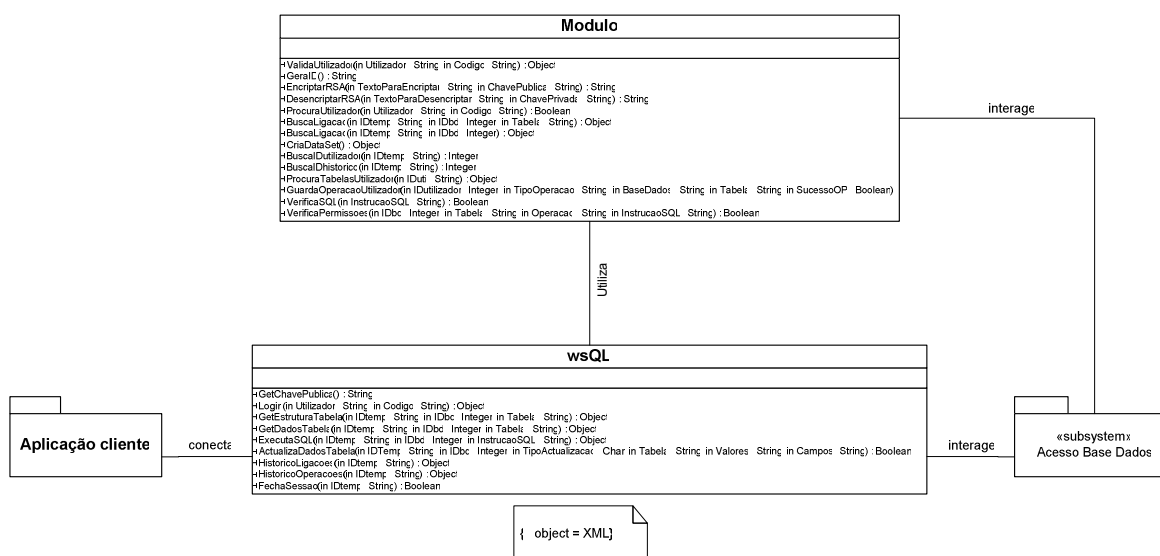


Figura 5.5 – Modelo de classes do sistema wsQL

Classe wsQL

Representa as operações públicas do Web Service. Na prática é através desta classe que a aplicação cliente estabelece comunicação com o Web Service, ou seja, é ela que disponibiliza as operações que a aplicação cliente tem conhecimento através do documento WSDL. Para isso, a aplicação cliente usa uma instância desta classe.

Na tabela 5.2 descrevemos as operações desta classe, fazendo uma abordagem aos dados de entrada e aos dados de saída.

Tabela 5.2 – Descrição da classe wsQL

Operação	Parâmetros	Descrição
GetChavePublica		A aplicação wsQL devolve a chave pública para a aplicação cliente, que a usa para encriptar a comunicação com o wsQL. Esta operação não recebe parâmetros.
Login	Utilizador Codigo	Operação para a aplicação cliente proceder à sua validação perante a aplicação wsQL. Se a validação tiver sucesso será retornada à aplicação cliente uma lista com as bases de dados e tabelas a que a aplicação cliente tem acesso, com as respectivas permissões. Esta operação é invocada sempre que a aplicação cliente se liga ao wsQL. Esta operação recebe como parâmetros Utilizador e Codigo encriptado.
GetEstruturaTabela	IDtemp IDbd Tabela	Devolve a estrutura de uma tabela, desde que a aplicação cliente tenha permissão para tal. A aplicação cliente invoca esta operação sempre que precisa da estrutura de uma tabela. Esta operação recebe como parâmetros IDtemp, que representa o identificador temporário, IDbd, que é o identificador da base de dados e Tabela, que representa o nome da tabela da qual a aplicação cliente requer a estrutura. Todos estes parâmetros fazem parte da lista enviada quando é efectuado o Login.
GetDadosTabela	IDtemp IDbd Tabela	Devolve todos os registos de uma tabela, desde que a aplicação cliente tenha permissão para tal. Esta operação recebe os mesmos parâmetros da operação GetEstruturaTabela, tendo o mesmo significado.
ExecutaSQL	IDtemp IDbd InstrucaoSQL	A aplicação cliente pode executar um conjunto de comandos SQL numa determinada base de dados para a qual tenha permissão. Esta operação recebe como parâmetros IDtemp, IDbd e InstrucaoSQL que representa o comando SQL enviado pela aplicação cliente para o wsQL submeter à base de dados.

ActualizaDadosTabela	IDtemp IDbd TipoActualizacao Tabela Valores Campos	A aplicação cliente usa esta operação para efectuar a manutenção de registos (inserir, alterar e eliminar) numa determinada tabela. Para isso, além dos parâmetros IDtemp e IDbd, recebe também TipoActualizacao que indica o tipo de manutenção, podendo ter os valores “I”, “U” ou “D”. Tabela indica qual a tabela vai ser actualizada. Os parâmetros Valores e Campos assumem significados diferentes de acordo com a operação de manutenção.
HistoricoLigacoes	IDtemp	Devolve uma lista com o histórico de ligações de um utilizador com data de início e fim de ligação.
HistoricoOperacoes	IDtemp	Devolve uma lista com o histórico de operações de um utilizador com data e hora e operações efectuadas com indicação de sucesso.
FechaSessao	IDtemp	Fecha a ligação entre a aplicação cliente e a aplicação wsQL.

Classe Modulo

Esta classe é constituída por um conjunto de funcionalidades de auxílio à implementação das operações públicas da classe wsQL. Essas funcionalidades incluem tarefas como gerar o identificador temporário, validar utilizadores, encriptar dados, certificar permissões, e correcção de erros nas instruções SQL, entre outras.

Na tabela 5.3 temos a descrição das funcionalidades desta classe, com os respectivos parâmetros.

Tabela 5.3 – Descrição da classe Modulo

Operação	Parâmetros	Descrição
EncriptarRSA	TextoParaEncriptar ChavePublica	Devolve o texto encriptado utilizando a chave privada através do algoritmo RSA.
DesencriptarRSA	TextoParaDesencriptar ChavePrivada	Devolve o texto desencriptado aplicando a chave privada.
ProcuraUtilizador	Utilizador Codigo	Procura um utilizador
BuscaLigacao	IDtemp IDbd Tabela	Procura os dados necessários para efectuar a ligação ao sistema de gestão de bases de dados.
CriaDataSet		Cria a estrutura para a lista de informações a devolver quando o login é efectuado com sucesso.
BuscaIDutilizador	IDtemp	Em cada ligação é atribuído à aplicação cliente um identificador temporário e esta função permite obter os dados desse identificador.

BuscaIDhistorico	IDtemp	Permite obter os dados do identificador temporário, para efeitos de histórico.
ProcuraTabelasUtilizador	IDuti	Procura todas as tabelas que um determinado utilizador através do número atribuído internamente ao utilizador.
GuardaOperacaoUtilizador	IDutilizador TipoOperacao BaseDados Tabela SucessoOP	Guarda em histórico as informações de cada operação efectuada pela aplicação cliente. Essas informações incluem o tipo de operação, a base de dados, a tabela e o sucesso da mesma.
ValidaUtilizador	Utilizador Codigo	Permite informar acerca da validade do utilizador perante o sistema.
GeraID		Gera aleatoriamente uma cadeia de 20 caracteres para identificar cada ligação por parte da aplicação cliente.
VerificaSQL	InstrucaoSQL	Tem por objectivo analisar e corrigir todas as instruções SQL enviadas pela aplicação cliente ao wsQL, isto é, verifica e corrige erros de sintaxe.
VerificaPermissoes	IDbd Tabela Operacao InstrucaoSQL	Verifica se a aplicação cliente tem permissão para invocar determinada operação sobre uma tabela. Verifica também se a aplicação cliente tem permissão para executar os comandos contidos na instrução SQL que enviou.

Devido à dependência do subsistema do Web Service wsQL relativamente aos outros pacotes, a sua construção será abordada apenas no ponto 5.5.

5.3 Subsistema de Acesso a Dados

Este subsistema de Acesso a Dados que tem a sua concretização na camada de dados da arquitectura do sistema wsQL, é responsável pelas operações de acesso e manutenção de bases de dados dos sistemas de gestão de bases de dados.

Esta camada entra em acção sempre que a aplicação cliente invoca uma operação do wsQL que envolva acesso a base de dados, através da camada de aplicação. Esta, depois de validar a operação requerida pela aplicação cliente, envia os parâmetros à camada de dados para que efectue a operação no sistema de gestão de bases de dados, numa determinada base de dados.

5.3.1 Modelo de classes

Para cumprir o descrito anteriormente, e também para cumprir o objectivo de flexibilidade e para permitir a inclusão de novos sistemas de gestão de bases de dados, este componente deve ter a capacidade de lidar com vários sistemas de gestão de bases de dados e ser estruturado para permitir a escalabilidade do sistema.

As classes identificadas para a camada de dados são:

- Classe SGBD;
- Classe AcessoBaseDados.

Na figura 5.6 temos o diagrama de classes que representa o modelo da implementação da camada de dados.

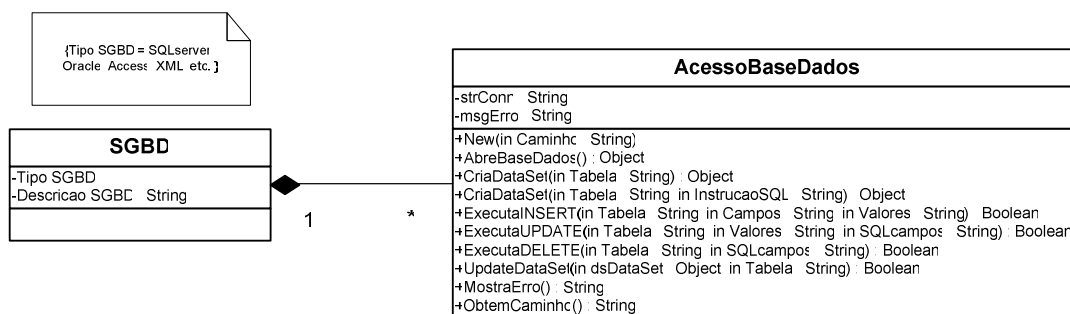


Figura 5.6 – Modelo de classes da camada de dados

Classe SGBD

Esta classe representa um sistema de gestão de bases de dados. Assim, para cada sistema de gestão de bases de dados, é criada uma classe com o conjunto de operações da classe *AcessoBaseDados*, com a codificação específica para aceder a esse sistema de gestão de bases de dados.

Classe AcessoBaseDados

Esta classe é formada por todas as operações que fornecem as funcionalidades para aceder a uma base de dados; seleccionar, inserir, alterar e eliminar registos. Cada uma destas operações é implementada de acordo com as regras de cada um dos sistemas de gestão de bases de dados com que o sistema wsQL pode comunicar.

Na tabela 5.4 podemos analisar cada uma das operações que a classe implementa, para cada um dos sistemas de gestão de bases de dados.

Tabela 5.4 – Descrição da classe AcessoBaseDados

Operação	Parâmetros	Descrição
New	[Caminho]	Representa o construtor da classe. Sempre que a camada de aplicação cria um objecto desta classe, é criada uma <i>string</i> que representa a forma de ligação à base de dados. Esta operação tem duas implementações. Uma sem qualquer parâmetro em que o caminho da base de dados é encontrado através da operação ObtemCaminho. Outra implementação recebe como parâmetro o caminho da base de dados.
AbreBaseDados		Cria uma conexão com a base de dados utilizando a <i>string</i> criada no construtor.
CriaDataSet	Tabela [InstrucaoSQL]	Permite seleccionar registos de uma determinada tabela. Tem duas implementações através do conceito de sobrecarga de operações, uma em que recebe apenas o parâmetro Tabela e, neste caso, são seleccionados todos os registos, e outra em que recebe também uma InstruçãoSQL que pode conter, por exemplo, uma selecção com condições.
ExecutaINSERT	Tabela Campos Valores	Permite inserir registos numa determinada tabela. Recebe como parâmetros Tabela, Campos com dados e Valores para esses campos.
ExecutaUPDATE	Tabela Campos SQLcampos	Permite actualizar registos numa determinada tabela. Recebe como parâmetros Tabela, Campos a alterar com os respectivos valores e o parâmetro SQLcampos que indica quais os campos afectados pela alteração.
ExecutaDELETE	Tabela SQLcampos	Permite eliminar registos de uma determinada tabela. Recebe como parâmetros Tabela, SQLcampos que indica quais os campos afectados pela eliminação.
UpdateDataSet	dsDataSet Tabela	Permite actualizar todos os registos ou um conjunto de registos de uma tabela.
ObtemCaminho		Obtem o caminho da base de dados a partir de um ficheiro de configuração wsQL.XML.
MostraErro		Permite visualizar qualquer erro que ocorra em qualquer uma das operações anteriores.

5.3.2 Construção da camada de dados

O componente que implementa a camada de dados foi construído de forma a poder cumprir vários objectivos: o primeiro era ser autónomo, de modo a permitir a programação e teste sem afectar todo o sistema; o segundo era ser capaz de aceder a vários sistemas de

gestão de bases de dados; e o terceiro era a qualquer momento poder acomodar novos sistemas de gestão de bases de dados.

Nesse sentido, a codificação consistiu em criar um projecto, que na plataforma .NET assume o nome de *ClassLibrary*. Nesse projecto temos uma classe para cada sistema de gestão de bases de dados e em cada classe codificamos as operações, segundo as regras de acesso a cada sistema de gestão de bases de dados. A codificação presente no projecto, depois de compilada, concretiza-se num ficheiro físico de biblioteca dinâmica, com extensão DLL.

No exemplo 5.1 podemos ver um pequeno excerto da codificação do componente da camada de acesso a dados.

Exemplo 5.1 – Excerto do código do componente da camada de dados

Namespace AcessoDados

```
Public Class clsAcessoDadosSQL
    ...
    Public Function AbreBaseDados() As SqlConnection

        Dim Conn As SqlConnection

        Conn = New SqlConnection(strConn)

        Try
            Conn.Open()
        Catch myErr As System.Exception
            msgErro = "Servidor ou Base de Dados Inexistente!"
            msgErroOriginal = myErr.ToString
        End Try

        Return Conn

    End Function
    ...
End Class

Public Class clsAcessoDadosOLEDB
    ...
    Public Function AbreBaseDados() As OleDbConnection

        Dim Conn As OleDbConnection

        Conn = New OleDbConnection(strConn)

        Try
            Conn.Open()
        Catch myErr As System.Exception
            msgErro = "Servidor ou Base de Dados Inexistente!"
            msgErroOriginal = myErr.ToString
        End Try

        Return Conn

    End Function
    ...
End Class
```

```
Public Class clsAcessoDadosORACLE
    ...
    Public Function AbreBaseDados() As OracleConnection

        Dim Conn As OracleConnection

        Conn = Ligacao()

        Try
            Conn.Open()
        Catch myErr As System.Exception
            msgErro = "Servidor ou Base de Dados Inexistente!"
            msgErroOriginal = myErr.ToString
        End Try

        Return Conn

    End Function
    ...
End Class

End Namespace
```

Nesta primeira versão do projecto codificámos as operações que permitem seleccionar, inserir, alterar e eliminar registos nos seguintes sistemas de gestão de bases de dados:

- SQL Server;
- Access;
- Oracle.

Como já referido, a arquitectura do sistema permite adicionar novos sistemas de gestão de bases de dados, sem que isso afecte o sistema visto de uma forma global, ou seja, a parte do servidor e a parte do cliente. Isto porque, na parte do servidor, as operações disponibilizadas pelo Web Services e o acesso aos dados estão em camadas separadas, portanto, autónomas. Uma vez que a aplicação cliente depende do servidor, é muito importante que as operações públicas não sofram modificações. Na parte do cliente, para a aplicação cliente, o sistema de gestão de bases de dados é totalmente transparente, isto é, qualquer que ele seja, a forma de o aceder e manipular concretiza-se através da mesma operação.

Portanto, para procedermos à codificação do acesso a outros sistemas de gestão de base de dados, basta acrescentar uma nova classe para cada sistema de gestão de base de dados ao componente de acesso a dados, compilar o componente e substituir o antigo. Desta forma, a organização fornecedora do Web Service, passa a poder disponibilizar acesso a base de dados desses sistemas de gestão de base de dados. As operações públicas do Web Service permanecem inalteráveis.

Assim, a aplicação cliente, sem ser necessário qualquer codificação adicional, quando se conectar novamente ao Web Service, recebe mais uma base de dados na lista com que pode interagir e, independentemente da base de dados, utiliza a mesma operação do Web Service para executar determinada tarefa.

5.4 Subsistema de Gestão da Aplicação

Este subsistema deve permitir ao responsável da organização fornecedora dos serviços efectuar a gestão e o controlo da aplicação wsQL. Para isso, foi criada uma aplicação, com uma interface gráfica, que disponibiliza um conjunto de opções que auxilia nas tarefas de gestão e controlo do Web Service wsQL.

A aplicação foi construída para ser utilizada através da *Web*, recorrendo a um simples *browser*, para permitir a gestão e o controlo do sistema remotamente. Este sistema de informação deve ser apoiado por uma base de dados persistentes, para guardar a informação gerada na aplicação, ou seja informação acerca dos utilizadores, das bases de dados partilhadas, das tabelas disponíveis para cada utilizador com os respectivos privilégios destes sobre estas, e os históricos de ligações da aplicação cliente.

5.4.1 Modelo de classes

Para satisfazer os requisitos identificados para o gestor da aplicação foram definidas as classes da figura 5.7.

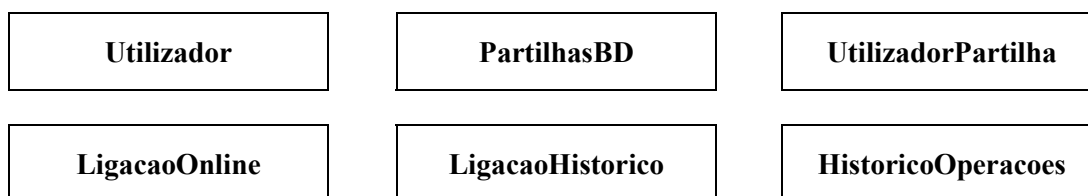


Figura 5.7 – Identificação das classes

Utilizador – representa um utilizador, isto é, uma aplicação cliente, que tem permissão para aceder e utilizar o wsQL. Cada Utilizador é caracterizado por um identificador único, um nome, um código de acesso e o nome da empresa.

PartilhasBD – representa uma base de dados disponível para partilhar entre os utilizadores. Cada PartilhaBD é caracterizada por uma descrição, pelo tipo de base de dados (SQL Server, Oracle, MDB, XML, etc.) e pela definição dos parâmetros necessários ao acesso para cada tipo de base de dados.

UtilizadorPartilha – representa a permissão de acesso a uma base de dados/tabela para um determinado utilizador. Cada UtilizadorPartilha é caracterizado pela base de dados/tabela a que tem acesso e pelas permissões de selecção, inserção, alteração e eliminação de registos.

LigacaoOnLine – representa uma ligação de um determinado utilizador, que num dado momento está a usar o wsQL. Cada LigacaoOnLine é caracterizada por um utilizador e um identificador único para a ligação.

LigacaoHistorico – representa cada ligação efectuada por uma determinada aplicação cliente. Cada LigacaoHistorico é caracterizada por um utilizador, a data/hora de início da ligação e data/hora de fim de ligação.

HistoricoOperacao – representa cada operação efectuada por uma determinada aplicação cliente. Cada HistoricoOperacao é caracterizada por um utilizador, a operação efectuada, data/hora da operação, base de dados utilizada, tabela e sucesso da operação.

Depois de identificadas todas as classes, elaborámos um diagrama geral que inclui todas as classes que suportam a informação necessária à aplicação de gestão. Esse diagrama pode ser analisado na figura 5.8.

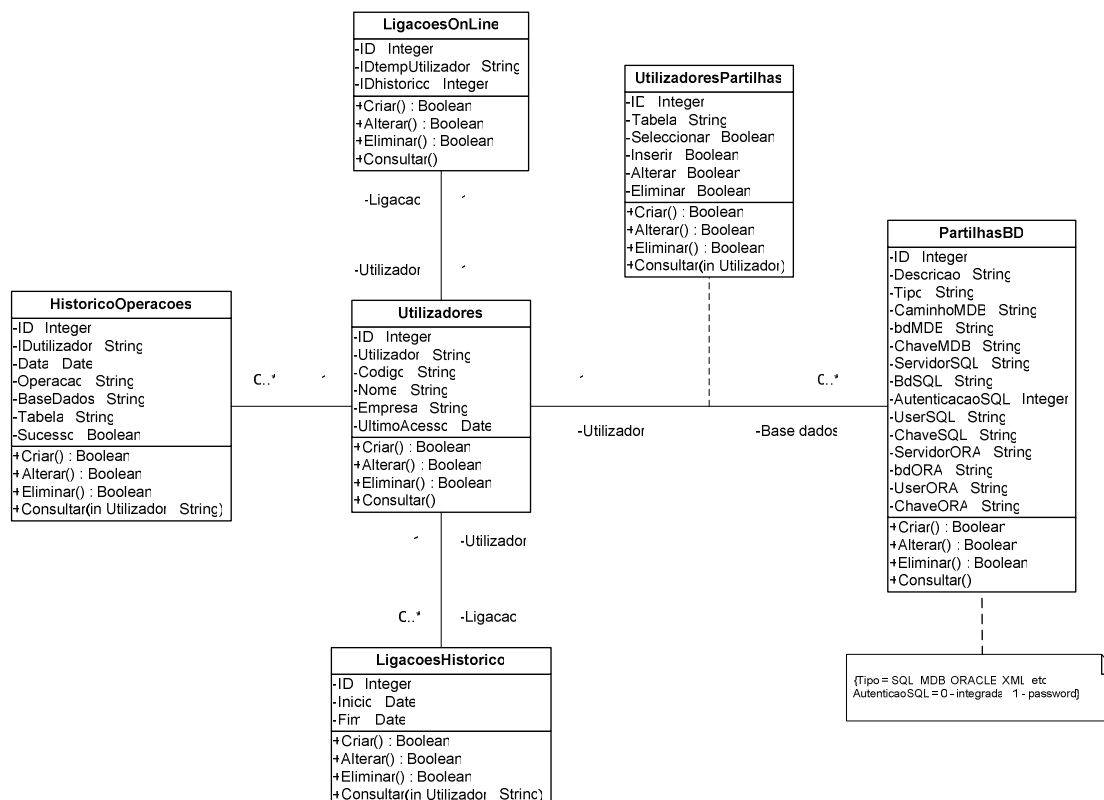


Figura 5.8 – Modelo de classes da aplicação de gestão do sistema

5.4.2 Modelo relacional

Como o diagrama de classes foi elaborado com a perspectiva de modelação da estrutura de uma base de dados, seguindo as regras de transposição a partir do modelo de classes, obtemos o modelo relacional que pode ser visto na figura 5.9.

Utilizador(ID, Utilizador, Nome, Codigo, Empresa, UltimoAcesso)

PartilhasBDs(ID, Descricao, Tipo, CaminhoMDB, bdMDB, ChaveMDB, ServidorSQL, bdSQL, AutenticacaoSQL, UserSQL, ChaveSQL, ServidorORA, bdORA, UserORA, ChaveORA)

UtilizadoresPartilhas(ID, IDbd, IDutilizador, Tabela, Seleccao, Inserir, Actualizar, Apagar)

LigacoesOnLine(ID, IDutilizador, IDutilizadorTemp, IDhistorico)

LigacoesHistorico(ID, IDutilizador, Inicio, Fim)

HistoricoOperacoes(ID, IDutilizador, Data, Operacao, BaseDados, Tabela, Sucesso)

Figura 5.9 – Modelo relacional

Na figura 5.10 pode ser visto o diagrama relacional criado a partir da descrição das tabelas apresentada na figura 5.9. Esse diagrama foi elaborado orientado para utilização no sistema de gestão de bases de dados SQL Server.

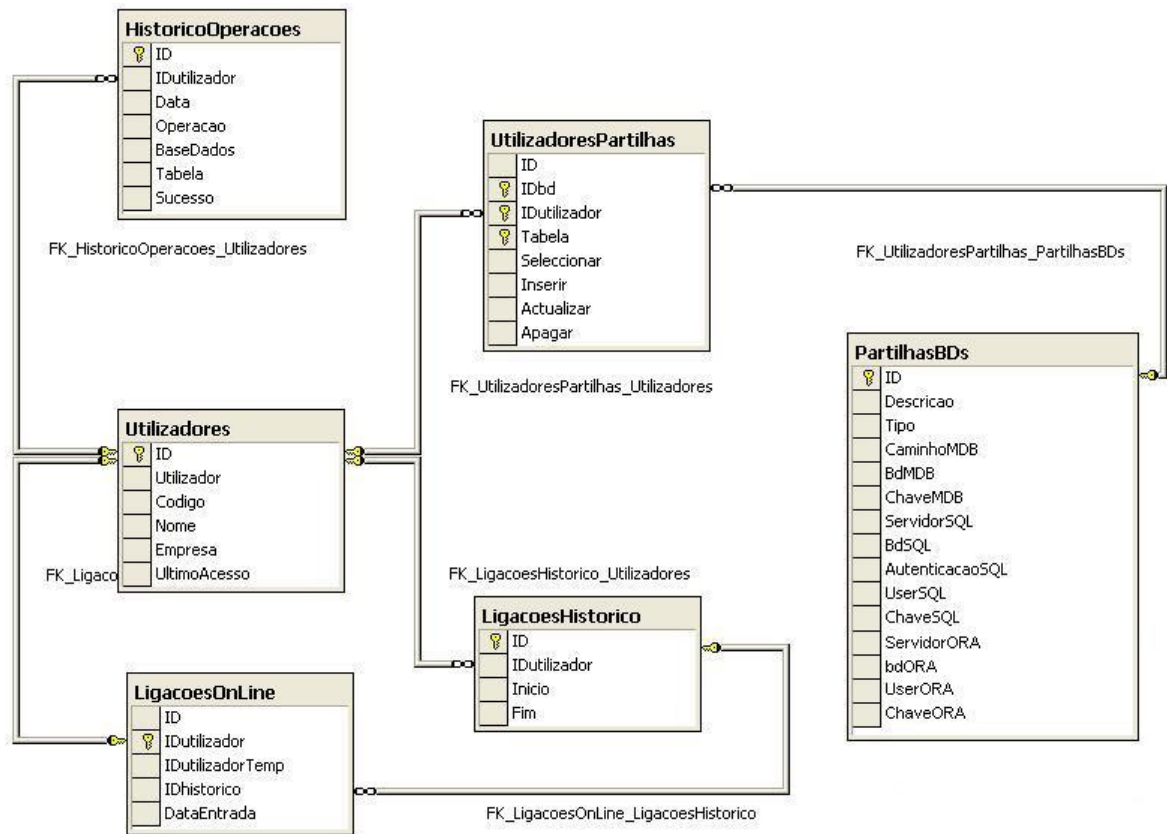


Figura 5.10 – Diagrama do modelo relacional da aplicação de gestão do wsQL

No anexo B, podemos analisar a definição de cada uma das tabelas apresentadas na figura 5.10. A definição inclui, para cada campo, o nome, o tipo de dados, o comprimento e as chaves primárias e secundárias.

5.4.3 Construção da aplicação de Gestão da Aplicação

A principal função desta aplicação é permitir ao gestor da aplicação wsQL executar as funcionalidades identificadas na fase de análise de todo o sistema, que podem ser subdivididas em: configuração, gestão, monitoração e controlo.

Considerando o modelo de desenvolvimento de três camadas, para esta aplicação, todas as três camadas, i.é, camada de apresentação, aplicação e dados são implementadas no lado do servidor.

Na figura 5.11 temos a interface gráfica da aplicação de gestão do sistema wsQL, com as funcionalidades disponíveis.



Figura 5.11 – Interface gráfica da aplicação de gestão do sistema wsQL

Através da aplicação apresentada anteriormente, o gestor do sistema wsQL pode efectuar as seguintes tarefas:

- **Partilha de base dados** – acção em que o gestor define quais as bases de dados da organização que poderão ser disponibilizadas a aplicações clientes internas ou de outras organizações externas. Para cada base dados é guardada na tabela PartilhasBDs uma breve descrição, o tipo de sistema de gestão de base de dados, e os parâmetros necessários ao acesso à base de dados.
- **Gestão de utilizadores** – gestão dos utilizadores que podem aceder ao sistema wsQL através de aplicações. Para isso, a cada utilizador é associado um código de acesso.
- **Partilha de bases de dados por utilizador** – nesta acção, o gestor do sistema define para cada utilizador quais as tabelas a que este pode aceder através das operações do Web Service wsQL, assim como as permissões de selecção, inserção, alteração e eliminação de registos.

- **Visualizar as ligações *online*** – o gestor através desta acção pode monitorar quais os utilizadores que têm aplicações conectadas e que operações do wsQL estão a usar no momento.
- **Consultar histórico de ligações** – através desta opção, o gestor pode consultar o histórico de ligações e operações utilizadas, a que bases de dados e tabelas foram aplicadas as operações e as instruções SQL submetidas. Pode também analisar se as operação requisitadas obtiveram sucesso.
- **Invocar o Web Service wsQL** – através desta acção o gestor pode certificar se o Web Service está operacional.

Toda a informação gerada nesta aplicação tem um carácter persistente, e por isso é guardada na base de dados definida na figura 5.10 e que foi implementada no SQL Server.

Assim, para o bom funcionamento do sistema wsQL, faz também parte da aplicação de gestão a respectiva base de dados. Na figura 5.12 podemos analisar um diagrama que representa os processos de instalação da aplicação de gestão na organização fornecedora dos serviços do wsQL.

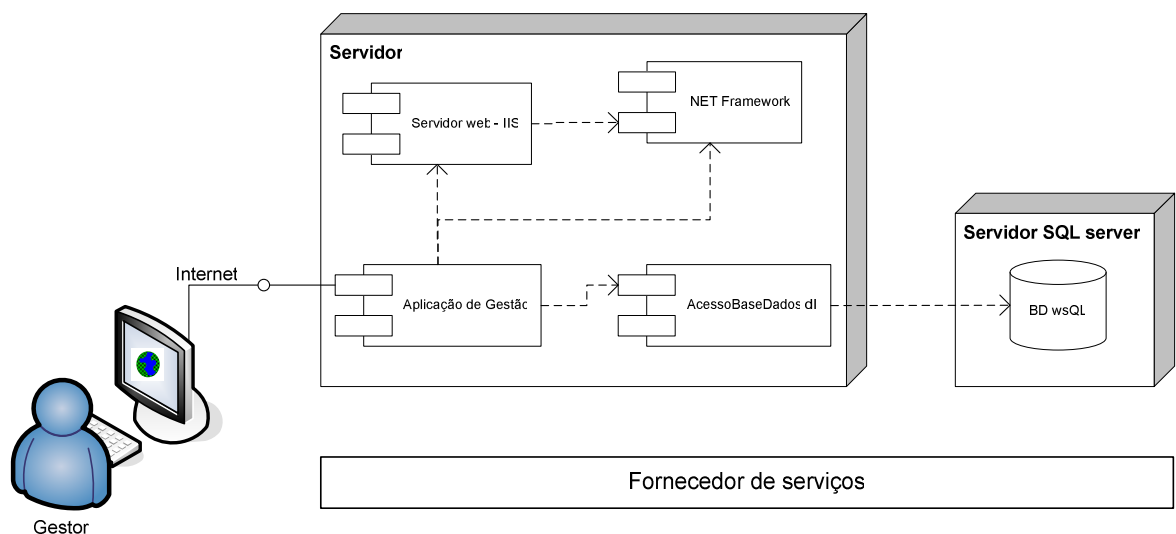


Figura 5.12 – Diagrama de instalação da aplicação de gestão dos sistema wsQL

Este subsistema tira partido do desenvolvimento de componentes autónomos, como se pode ver na figura 5.12: a aplicação de gestão do sistema também usa o mesmo componente *AcessoBaseDados* que foi criado inicialmente para ser usado pelo Web Service.

A informação criada na aplicação de gestão tem um papel fundamental em todo o sistema, pois é com base nesta informação que o wsQL toma decisões e as operações podem efectuar as funcionalidades para que foram programadas. As decisões incluem aceitar ou rejeitar a conexão da aplicação cliente. No caso de aceitação, deve descobrir quais as bases de dados e tabelas disponíveis para essa aplicação cliente, para lhe fazer chegar a lista. Depois, sempre que a aplicação cliente invoque uma operação, é com base na informação da base de dados de suporte ao sistema, que o Web Service verifica as permissões da aplicação cliente para efectuar tal operação.

5.5 Construção do Web Service wsQL

A construção do Web Service wsQL é abordada apenas nesta secção, devido ao facto de este depender dos outros dois subsistemas, o de acesso a sistemas de gestão de bases de dados e o de gestão da aplicação. O Web Service usa uma instância do primeiro para poder utilizar as operações para manipulação de bases de dados e, do segundo, usa a informação gerada nesse subsistema, para verificar a permissão de acesso às suas operações e sua invocação por parte da aplicações cliente.

Como mencionado na secção 5.1.1 a construção do Web Service, pode ser subdividida em duas partes: a primeira consiste em definir quais as operações públicas, bem como para cada operação quais os parâmetros de entrada e quais os dados de saída; e a segunda consiste em programar as funções que implementam as funcionalidades das operações do Web Service, assim como também um conjunto de funções que permitem controlar as aplicações cliente, tais como validar utilizadores, verificar permissões de aceder a bases de dados e tabelas, validar e corrigir instruções SQL enviadas pela aplicação cliente e ainda outras funções para gerar o identificador temporário associado a cada ligação de uma aplicação cliente, e guardar em histórico toda a informação acerca de cada operação invocada pela aplicação cliente.

5.5.1 Operações do Web Service wsQL

Consequência de toda a análise efectuada, da modulação do sistema, assim como da construção da camada de dados e da aplicação de gestão, chegámos à fase da transposição

dos modelos definidos anteriormente para construir o núcleo do sistema, o Web Service propriamente dito. Basicamente, era preciso juntar todas estas peças, isto é fazer a integração de todos os subsistemas de forma a criar um sistema completo e unitário.

Na figura 5.13 apresentamos as operações do Web Service wsQL, colocadas numa página Web gerada pela plataforma .NET, para visualização e teste das operações de um Web Service.

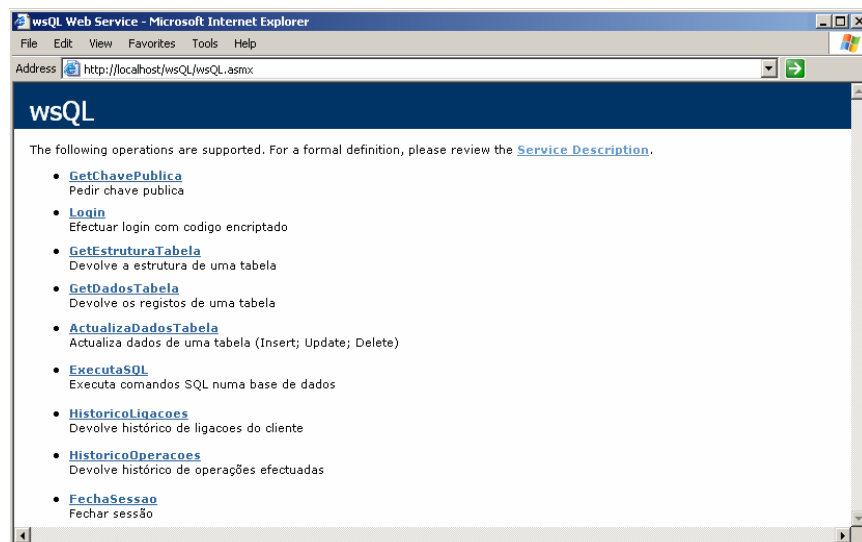


Figura 5.13 – Operações públicas do Web Service wsQL

Na plataforma .NET a criação de um Web Service realiza-se através de um projecto *ASP.NET Web Service* e de uma das linguagens de programação disponíveis.

O ficheiro principal de um projecto deste tipo tem extensão ASMX e a primeira linha possui a seguinte directiva:

```
<%@ WebService Language="vb" Codebehind="wsQL.asmx.vb" Class="wsQL.wsQL" %>.
```

Basicamente, esta directiva indica que a programação efectuada é para ser usada por um Web Service, que a linguagem utilizada neste caso é o VB.NET, o nome da classe que implementa o Web Service e opcionalmente pode ter, como acontece no exemplo, o nome do ficheiro independente, *wsQL.asmx.vb*, onde são codificadas as operações do Web Service; caso contrário a codificação será efectuada no ficheiro ASMX.

O ficheiro de codificação do Web Service começa por estabelecer as referências aos *namespaces* necessários para o Web Service, como a seguir se indica:

```
Imports System.Web.Services
Imports System.Security.Cryptography
```

Neste caso, as referências são às bibliotecas de classes de Web Services e à biblioteca de classes de criptografia da plataforma .NET.

Depois, temos a declaração da classe que representa o Web Service, ficando como uma subclasse de `System.Web.Services.WebService`, através de:

```
<System.Web.Services.WebService(Namespace:="http://fe.up.pt/wsQL/wsQL",
Description:="Web Service Query Language",Name:="wsQL")> _
Public Class wsQL
```

No exemplo, é definido um *namespace*, uma descrição e um nome para o Web Service.

Após a criação formal do Web Service, podemos agora criar as operações que constituem o Web Service, através do atributo `WebMethod`, como se mostra a seguir:

```
<WebMethod(Description:="Pedir chave publica")> _
Public Function GetChavePublica() As String
    ...
End Function
```

Aqui temos a definição da operação *GetChavePublica* que não recebe qualquer parâmetro e retorna uma *string* à aplicação que a invocar. Para cada operação a disponibilizar o atributo `WebMethod` deve obrigatoriamente ser colocado na definição dessa operação. No exemplo 5.2 apresentamos um excerto da codificação do Web Service `wsQL` com um dos métodos, *GetChavePublica*.

Exemplo 5.2 – Exemplo de codificação de um Web Service

```
1 Imports System.Web.Services
2 Imports System.Security.Cryptography
3
4 <System.Web.Services.WebService(Namespace:="http://fe.up.pt/wsQL/wsQL")> _
5 Public Class wsQL
6     Inherits System.Web.Services.WebService
7
8     Dim objDadoswsQL As New AcessoDados.clsAcessoDadosSQL
9     Private Shared InfoChavePublica As String
10    Private Shared InfoChavePrivada As String
11
12    <WebMethod(Description:="Pedir chave publica")> _
13    Public Function GetChavePublica() As String
14
15        If (InfoChavePrivada = Nothing) Then
16
17            Dim CSPparam As CspParameters
18            Dim RSA As RSACryptoServiceProvider
19
20            CSPparam = New CspParameters
21            CSPparam.Flags = CspProviderFlags.UseMachineKeyStore
22            RSA = New RSACryptoServiceProvider(CSPparam)
23
24            InfoChavePublica = RSA.ToXmlString(False)
25            InfoChavePrivada = RSA.ToXmlString(True)
26
27        End If
28
29        Return InfoChavePublica
30
31    End Function
32    ...
33 End Class
```

No exemplo anterior temos a criação do Web Service wsQL e também temos definida a título de exemplo uma das operações ou método disponibilizadas pelo wsQL. O objectivo da operação definida no exemplo anterior é criar um par de chaves, a pública e a privada, e devolver à aplicação cliente a chave pública. Esta é a primeira operação que a aplicação cliente deve invocar para obter a chave pública, que depois deve utilizar para encriptar os dados enviados.

Na linha 8 é criada uma instancia da classe *clsAcessoDadosSQL* da camada de dados *AcessoDados* para, desta forma, o Web Service poder usar todas as funcionalidades disponíveis na camada de dados.

Depois de compilado o Web Service concretiza-se em dois ficheiros físicos, um é biblioteca dinâmica, com extensão DLL, e outro com extensão ASMX.

A figura 5.14 demonstra a etapa de desenvolvimento do Web Service e a etapa de publicação do Web Service num servidor *Web*, para que fique disponível para ser usado por aplicações cliente remotas, via protocolos da Internet.

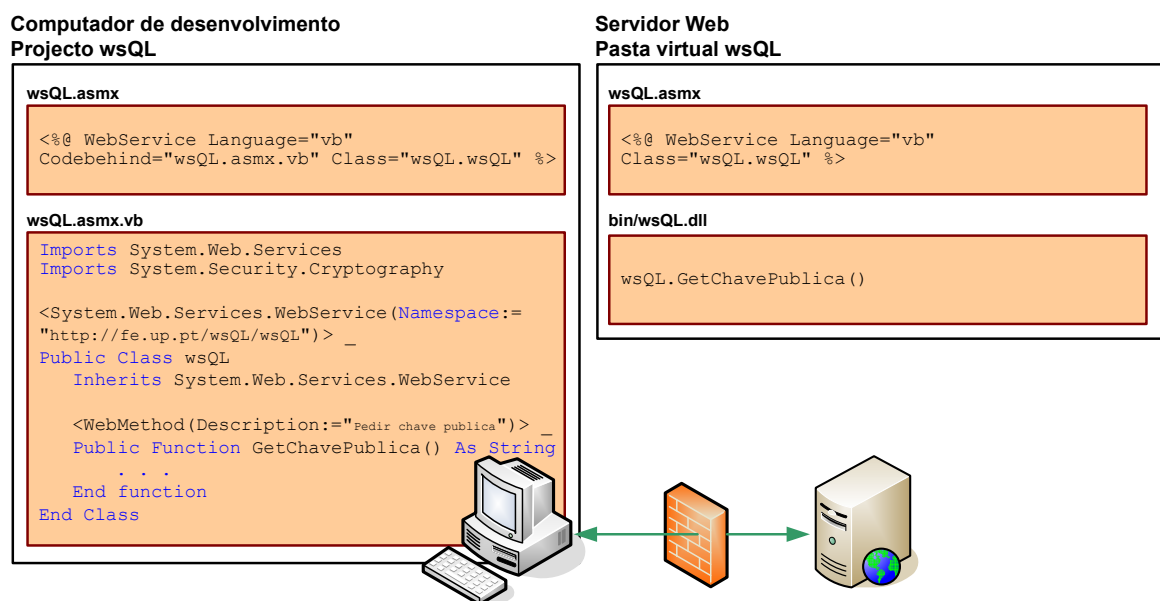


Figura 5.14 – Desenvolvimento e instalação do Web Service wsQL

A codificação de todas as operações do Web Service wsQL poder ser analisado no anexo C.

5.5.2 Funções auxiliares do Web Service wsQL

As funções auxiliares constituem um conjunto de funcionalidades codificadas em funções empacotadas no componente que designamos *Modulo*. Essas funções permitem que o Web Service possa controlar o acesso e o uso das suas operações por parte da aplicação cliente.

Sempre que uma aplicação cliente se conecta ao wsQL e invoca uma das suas operações, um conjunto de procedimentos que estão presentes neste componente, são accionados. Esses procedimentos incluem encriptação de dados, validação de utilizadores, verificar permissão de uso da operação, verificar permissão e correcção de instruções SQL, guardar a informação referente a cada operação e outros procedimentos que fazem com que o Web Service efectue as operações.

No exemplo 5.3 apresentamos um excerto da codificação do modulo com uma das funções, a função *EncryptarRSA*.

Exemplo 5.3 – Exemplo de codificação do modulo

```
Imports System.Security.Cryptography

Module Modulo

    Function DescriptarRSA(ByVal TextoParaDescriptar As String, _
                           ByVal ChavePrivada As String) As String

        Dim CSPparam As CspParameters
        Dim RSA As RSACryptoServiceProvider
        Dim BytesParaDescriptar As Byte()
        Dim TextoDescriptado As Byte()

        CSPparam = New CspParameters
        CSPparam.Flags = CspProviderFlags.UseMachineKeyStore
        RSA = New RSACryptoServiceProvider(CSPparam)
        RSA.FromXmlString(ChavePrivada)

        BytesParaDescriptar = System.Convert.FromBase64String(TextoParaDescriptar)
        TextoDescriptado = RSA.Decrypt(BytesParaDescriptar, False)

        Return System.Text.UnicodeEncoding.Unicode.GetString(TextoDescriptado)

    End Function

End Module
```

Esta função, que acima apresentamos, tem por objectivo converter para texto claro, isto é, texto legível, um texto encriptado com a chave pública do wsQL, utilizando para isso, a chave privada do wsQL. Esta função é invocada pela camada de aplicação do wsQL

sempre que a aplicação cliente envia dados encriptados com a chave pública, que obteve através da operação *GetChavePublica* do Web Service.

O algoritmo de encriptação presente no exemplo é o algoritmo assimétrico RSA. Cada letra da sigla representa o nome dos três indivíduos que contribuíram para o seu desenvolvimento, respectivamente Rivest, Shamir e Adleman [RSA, 2002].

5.6 Instalação do sistema wsQL

Uma organização que deseje disponibilizar acesso às suas bases dados a aplicações internas ou externas, através da *Web*, pode instalar o sistema wsQL. Este sistema é constituído por três subsistemas, o subsistema do Web Service, o subsistema de acesso aos sistemas de gestão de bases de dados e o subsistema da aplicação de gestão do sistema. O desenvolvimento desses subsistemas foi apresentado nas secções anteriores.

Para o funcionamento global do sistema, todos estes subsistemas terão de ser instalados na organização fornecedora dos serviços disponíveis no Web Service. No entanto, para se proceder à instalação de cada um destes subsistemas devem ser cumpridos determinados requisitos, que de seguida apresentamos.

Requisitos de instalação do subsistema do Web Service

Por definição, um Web Service é programado para ser acedido por outras aplicações, através de protocolos da Internet. Portanto, para que as funcionalidades do Web Service wsQL possam ser utilizadas por outras aplicações, é necessário que o ficheiro wsQL.asmx e o ficheiro wsQL.dll, que contêm as operações, sejam instalados numa pasta virtual de um servidor *Web*, tal e qual como uma tradicional página *Web*. Uma vez que utilizamos a plataforma .NET para o desenvolvimento deste sistema, o servidor *Web* deve ser o IIS, *Internet Information Server*, assim como também deve ser instalada a plataforma *.NET Framework* no servidor da organização.

Estas tarefas garantem que o Web Service wsQL esteja disponível para ser consumido, isto é, utilizado por outras aplicações. Opcionalmente, o gestor do sistema, pode publicar o Web Service em repositórios UDDI, com a finalidade de divulgar os serviços disponíveis no wsQL.

Requisitos de instalação do subsistema de acesso aos sistemas de gestão de base de dados

Uma cópia do componente *AcessoBaseDados.dll* deve ser instalada no servidor, na pasta do Web Service. A sua implementação também precisa das classes do componente *.NET Framework*.

Requisitos de instalação do subsistema da aplicação de gestão

Uma vez que se trata de uma aplicação *Web*, precisa de ser instalada numa pasta virtual do servidor *Web*. Uma cópia do componente *AcessoBaseDados.dll* também deve ser instalada na pasta da aplicação e esta aplicação também precisa das classes do componente *.NET Framework*.

Outra tarefa de instalação da aplicação de gestão consiste em colocar a base de dados de suporte à informação do sistema, no servidor do gestor de bases de dados. O gestor de bases de dados pode ser SQL Server ou então uma versão simplificada e gratuita, MSDE, *Microsoft Data Engine*.

Na figura 5.15 podemos analisar a organização dos ficheiros necessários para a instalação do sistema wsQL.

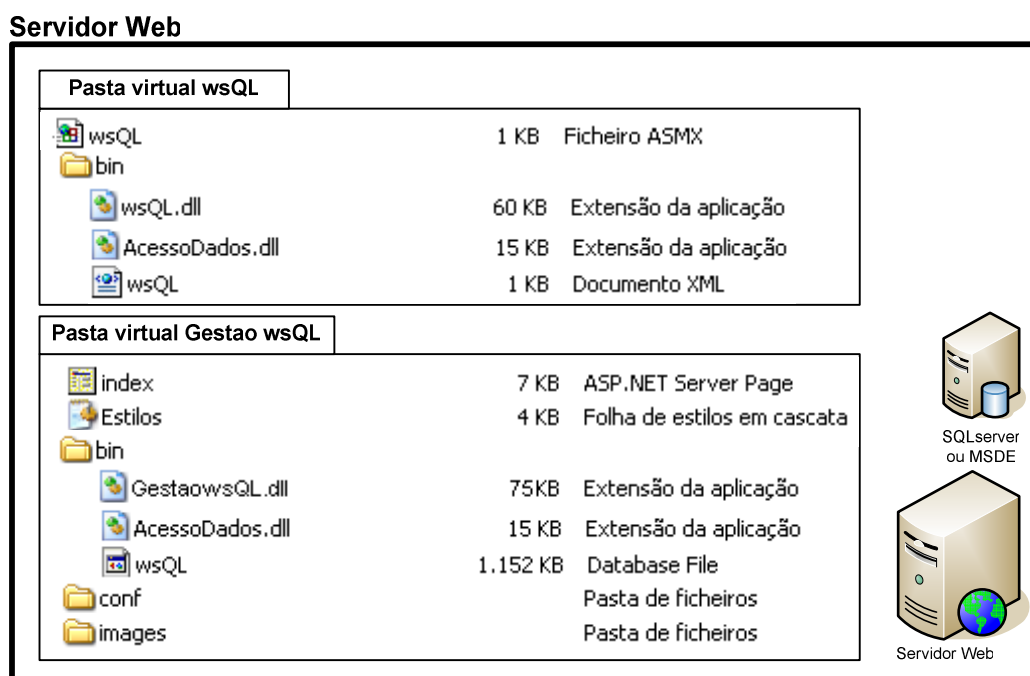


Figura 5.15 – Instalação dos ficheiros do sistema wsQL

Descritos os requisitos de instalação, na figura 5.16, apresentamos um diagrama de instalação do sistema wsQL, destacando os componentes instalados na parte do servidor e na parte do cliente, bem como a comunicação entre eles.

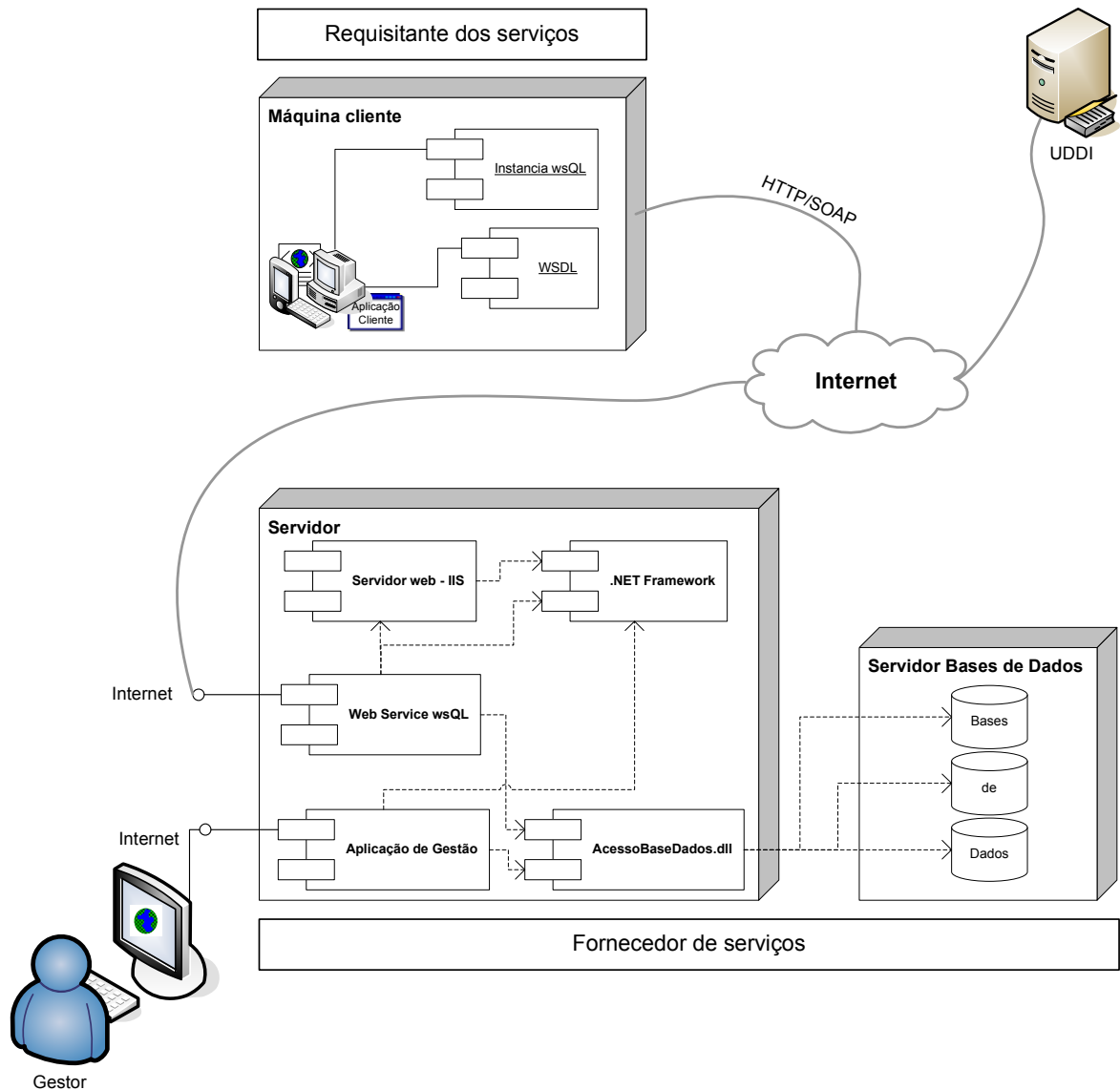


Figura 5.16 – Diagrama de instalação do sistema wsQL

Requisitos para utilização do Web Service pela aplicação cliente

Uma organização que pretenda utilizar os serviços disponíveis no sistema wsQL instalado internamente ou numa outra organização pode programar uma aplicação que efectue a ligação ao sistema wsQL e utilize as suas operações.

Para isso, deve obter o ficheiro WSDL no UDDI, ou através da organização fornecedora do wsQL. Depois, gerar um *class proxy*, basicamente, trata-se de uma

instância local do Web Service, que de facto é responsável pela comunicação da aplicação cliente com o Web Service. Esta classe pode ser gerada automaticamente ou manualmente, dependendo da plataforma de desenvolvimento usada. De seguida utilizar uma linguagem de programação e codificar a aplicação cliente, usando as operações do *proxy*.

Na figura 5.17 podemos observar os ficheiros necessários do lado do cliente para a programação da aplicação cliente, criados automaticamente através da plataforma .NET.

Aplicação cliente

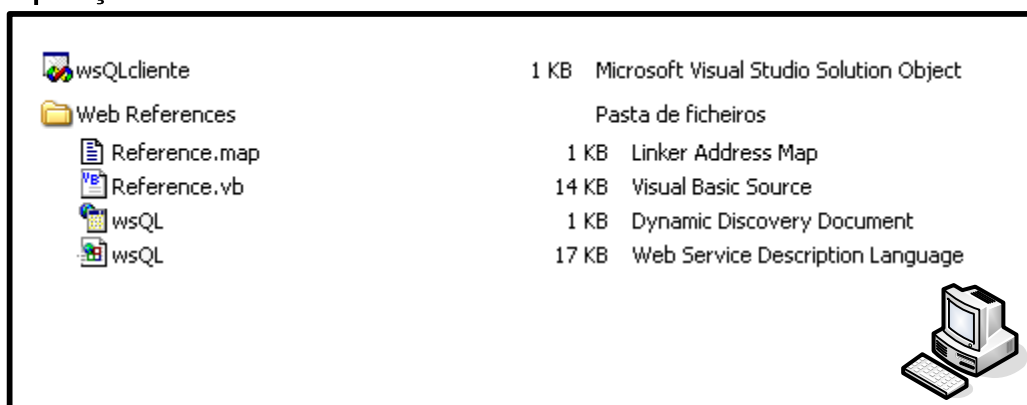


Figura 5.17 – Class proxy do wsQL

5.7 Validação do sistema wsQL

Concluída a construção do sistema wsQL – Web Service Query Language, chegou o momento de o validar.

Validar o sistema wsQL implica, inevitavelmente, logo à partida uma escolha, decidir por um cenário de aplicação real ou por um cenário simulado. A opção recaiu sobre uma cenário real.

5.7.1 Cenário de validação do sistema wsQL

A aplicação foi usada no seguinte cenário: uma instituição de ensino superior com quatro pólos geograficamente distantes: Bragança, Gaia, Leiria e Santarém. Gaia funciona como a sede, com privilégios distintos, e cada instituição pode aceder à base de dados do sistema de informação escolar de cada uma das três outras instituições. Gaia poderá fazer

também a manutenção das bases de dados. Na figura 5.18 podemos ver um diagrama ilustrativo do cenário descrito.

Para o processo de validação do sistema, descrevemos seguidamente as etapas fundamentais para que cada instituição pudesse disponibilizar acesso ao seu sistema de informação escolar através do sistema wsQL.

- **Etapas 1** – instalar o sistema wsQL no servidor *Web* de todas as quatro instituições, tal como foi descrito na secção 5.6;
- **Etapas 2** – o gestor do sistema de cada instituição, local ou remotamente, utilizando a aplicação de gestão, cria uma partilha de base de dados, indicando o caminho, o tipo de sistema de gestão de base de dados, e os parâmetros necessários à conexão, como por exemplo o código de acesso;
- **Etapas 3** – o gestor deve também criar um utilizador para identificar cada uma das outras instituições, recorrendo à aplicação de gestão;
- **Etapas 4** – o gestor deve criar um relação entre a base de dados e cada utilizador, isto é, para cada utilizador definir a que tabelas da base de dados pode aceder, bem como as permissões de selecção, inserção, alteração e eliminação de registos, utilizando também a aplicação de gestão.

Após a execução destas etapas em cada instituição, as aplicações cliente das outras instituições podem aceder ao sistema escolar uma das outras, utilizando para isso as operações do Web Service wsQL.

5.7.2 Aplicação cliente

O uso mais comum deste sistema visará certamente a integração de aplicações; no entanto, para proceder à validação do Web Service wsQL, criámos uma aplicação cliente de carácter geral, com o objectivo de usar e validar o sistema wsQL, em particular as operações do Web Service.

No primeiro écran dessa aplicação podemos definir o URL do Web Service a conectar, visualizar o documento WSDL e efectuar a ligação ao Web Service definido, através da operação *login* do Web Service. Na figura 5.19 apresentamos esse primeiro écran.

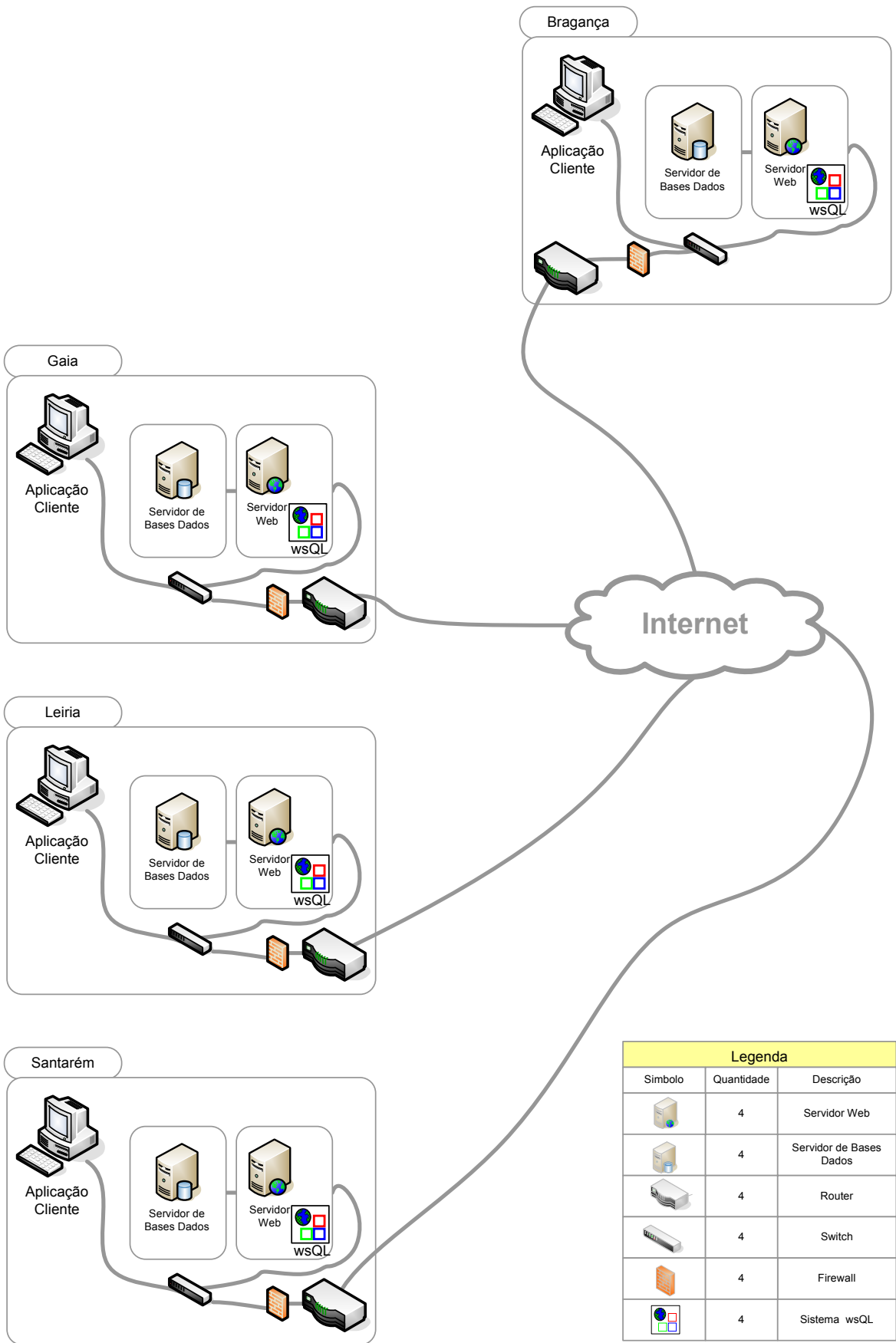


Figura 5.18 – Cenário para utilização do wsQL

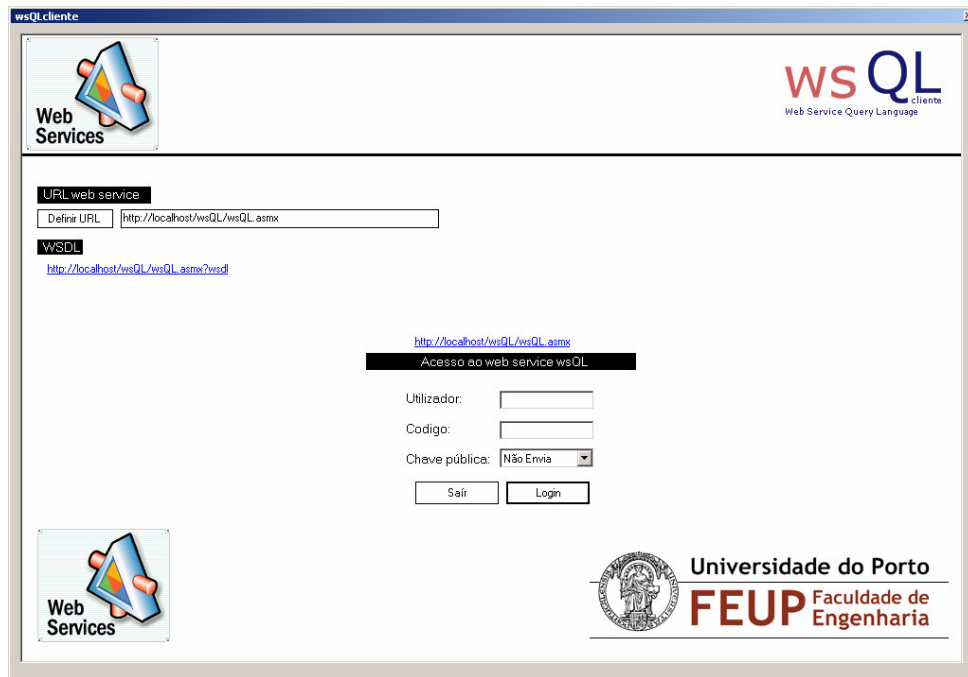


Figura 5.19 – Aplicação cliente

O utilizador especifica o nome de utilizador e o código de acesso, e pressiona o botão *login*, e depois disso a aplicação cliente invoca a operação *GetChavePublica* do Web Service, que retorna a chave pública, com que a aplicação cliente deve encriptar o código de acesso. Depois, a aplicação cliente invoca a operação *login* do Web Service enviando o nome do utilizador e o código de acesso encriptado como parâmetros da operação.

Caso a validação obtenha sucesso no Web Service, isto é, caso o utilizador exista no sistema wsQL a que estamos aceder e o código de acesso esteja correcto, o Web Service desse sistema retorna para a aplicação cliente uma lista de parâmetros sob a forma de um documento XML, como pode ser visto no exemplo 5.4.

Exemplo 5.4 – Lista de parâmetros retornada pelo Web Service

```
<?xml version="1.0" standalone="yes" ?>
<DataSet xmlns="http://fe.up.pt/wsQL/wsQL">
  <xs:schema id="NewDataSet" targetNamespace="http://fe.up.pt/wsQL/parametros.xsd"
    xmlns:mstns="http://fe.up.pt/wsQL/parametros.xsd"
    xmlns="http://tempuri.org/parametros.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    attributeFormDefault="qualified" elementFormDefault="qualified">
    <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:Locale="pt-PT" msdata:EnforceConstraints="False">
      <xs:complexType>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="Parametros">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="IDutilizador" type="xs:string" minOccurs="0" />
                <xs:element name="Nome" type="xs:string" minOccurs="0" />
                <xs:element name="Empresa" type="xs:string" minOccurs="0" />
                <xs:element name="UltimoAcesso" type="xs:dateTime" minOccurs="0" />
                <xs:element name="IDbd" type="xs:int" minOccurs="0" />
                <xs:element name="BaseDados" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>

```

```

        <xs:element name="Tabela" type="xs:string" minOccurs="0" />
        <xs:element name="Seleccionar" type="xs:boolean" minOccurs="0" />
        <xs:element name="Inserir" type="xs:boolean" minOccurs="0" />
        <xs:element name="Atualizar" type="xs:boolean" minOccurs="0" />
        <xs:element name="Apagar" type="xs:boolean" minOccurs="0" />
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>

<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
<NewDataSet xmlns="">
    <Parametros>
        <IDutilizador>OiEaXRl7h4neNSgvtkAa</IDutilizador>
        <Nome>JM</Nome>
        <Empresa>mtm.feup</Empresa>
        <UltimoAcesso>2005-04-01T13:23:08</UltimoAcesso>
    </Parametros>
    <Parametros>
        <IDbd>1</IDbd>
        <BaseDados>Ligacao SISLA</BaseDados>
        <Tabela>Alunos</Tabela>
        <Seleccionar>true</Seleccionar>
        <Inserir>true</Inserir>
        <Atualizar>true</Atualizar>
        <Apagar>false</Apagar>
    </Parametros>
    <Parametros>
        <IDbd>1</IDbd>
        <BaseDados>Ligacao SISLA</BaseDados>
        <Tabela>Notas</Tabela>
        <Seleccionar>true</Seleccionar>
        <Inserir>false</Inserir>
        <Atualizar>false</Atualizar>
        <Apagar>false</Apagar>
    </Parametros>
    <Parametros>
        <IDbd>2</IDbd>
        <BaseDados>Ligacao SGIS</BaseDados>
        <Tabela>Alunos</Tabela>
        <Seleccionar>true</Seleccionar>
        <Inserir>true</Inserir>
        <Atualizar>true</Atualizar>
        <Apagar>true</Apagar>
    </Parametros>
    <Parametros>
        <IDbd>2</IDbd>
        <BaseDados>Ligacao SGIS</BaseDados>
        <Tabela>Docentes</Tabela>
        <Seleccionar>true</Seleccionar>
        <Inserir>false</Inserir>
        <Atualizar>false</Atualizar>
        <Apagar>false</Apagar>
    </Parametros>
</NewDataSet>
</diffgr:diffgram>
</DataSet>

```

O documento XML anterior pode ser dividido em duas partes: na primeira parte está definido o esquema XML e na segunda parte estão definidos os dados. Assim, submetendo-se o documento XML a um *parser* XML podemos construir a estrutura de elementos, ou seja, os campos e a estrutura de dados que se podem ver respectivamente na tabela 5.5 e 5.6. Basicamente, tratam-se de estruturas que definem uma tabela, a tabela *Parametros*, com um conjunto de campos.

Tabela 5.5 - Estrutura de campos definidos no esquema XML do exemplo 5.4

Parâmetros	
Nome campo	Tipo campo
IDutilizador	string
Nome	string
Empresa	string
UltimoAcesso	dateTime
IDbd	int
BaseDados	string
Tabela	string
Seleccionar	boolean
Inserir	boolean
Actualizar	boolean
Apagar	boolean

Tabela 5.6 – Estrutura de dados resultante do documento XML do exemplo 5.4

IDutilizador	Nome	Empresa	UltimoAcesso	. . .
OiEaXRl7h4neNSgvtkAa	JM	mtm.FEUP	2005-04-01T13:23:08	. . .
				. . .
				. . .
				. . .
				. . .

. . .	IDbd	BaseDados	Tabela	Seleccionar	Inserir	Actualizar	Apagar
. . .							
. . .	1	Ligacao SISLA	Alunos	true	true	true	false
. . .	1	Ligacao SISLA	Notas	true	false	false	false
. . .	2	Ligacao SGIS	Alunos	true	true	true	true
. . .	2	Ligacao SGIS	Docentes	true	false	false	false

Nesta informação que é enviada pelo sistema wsQL encontramos dados que serão usados para controlar a ligação e dados relacionados com as bases de dados e tabelas disponíveis à aplicação cliente. Assim, no primeiro registo temos o identificador temporário, que identifica a presente conexão e que a aplicação cliente deve enviar sempre na invocação das operações. Temos também o nome do responsável e da organização requisitante do serviço, bem como um campo que indica a data do último acesso ao sistema wsQL.

A partir do segundo registo, temos todas as bases de dados e tabelas disponíveis à aplicação cliente. Em cada registo, temos campos que identificam a base dados na

organização fornecedora do acesso e a tabela com um conjunto de campos que indicam as permissões.

No exemplo apresentado, podemos ver claramente que a esta aplicação cliente estão disponíveis duas bases de dados, identificadas no servidor com os números 1 e 2, números que serão usados nas operações para identificar as bases de dados. Na aplicação cliente, as bases de dados são identificadas através de uma descrição no campo *BaseDados*. Para cada tabela, temos também a informação das permissões de selecção, inserção, alteração e eliminação de registos.

Salientamos o facto de não termos qualquer referência ao sistema de gestão de base de dados ao qual cada base de dados pertence. Para a aplicação cliente, essa questão é transparente, ou seja, independentemente do sistema de gestão de bases de dados, a aplicação cliente usa a mesma operação para requisitar a mesma funcionalidade. A complexidade inerente ao tratamento dos vários sistemas de gestão de bases de dados é da inteira responsabilidade do Web Service.

Com a informação extraída dessa tabela, podemos construir o écran da figura 5.20. Este protótipo de uma aplicação cliente permite que o utilizador possa usar a aplicação de forma fácil e prática.

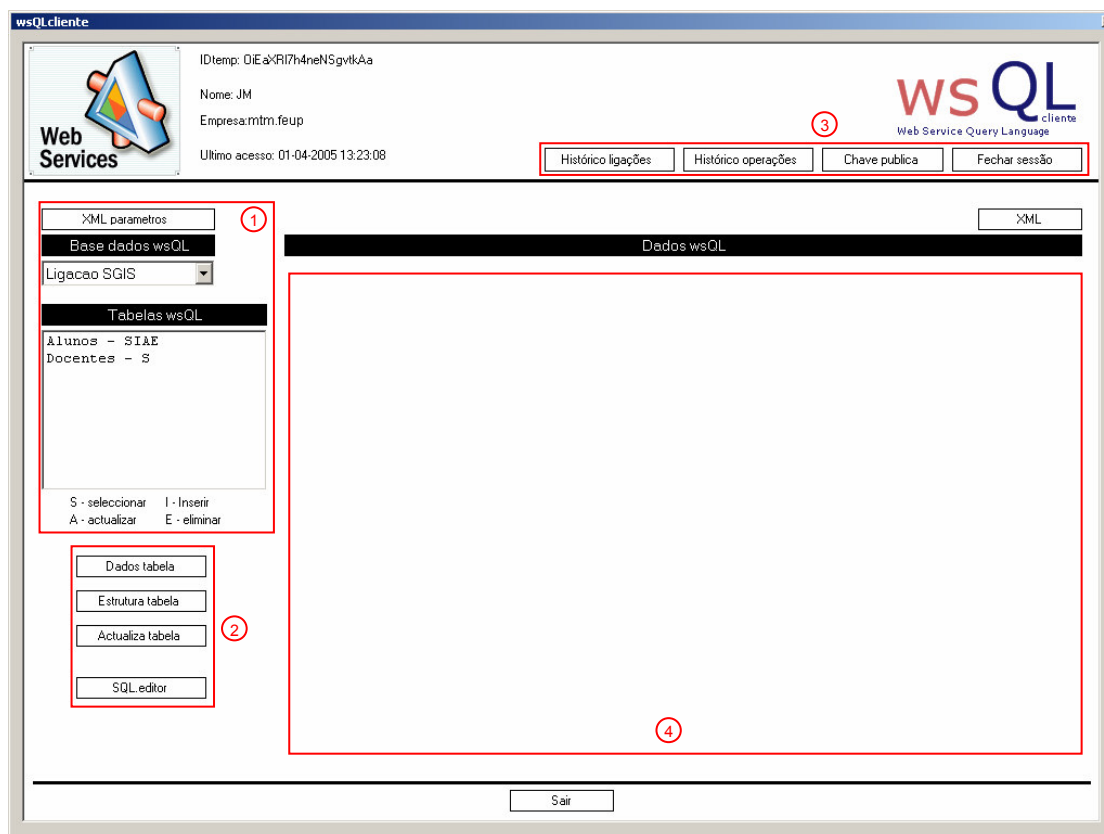


Figura 5.20 – Camada de apresentação do Web Service

Utilizando a interface apresentada na figura 5.20, podemos invocar e testar todas as operações do Web Service wsQL.

Na área identificada com o número 1, temos um botão que permite visualizar o documento XML que contém a lista de parâmetros, (exemplo 5.4). A caixa de combinação “Base dados wsQL” contém todas as bases dados disponíveis à aplicação cliente, e a lista “Tabelas wsQL” contém as tabelas de cada uma das bases dados da caixa de combinação anterior. Estes itens são povoados com os dados extraídos da lista de parâmetros que anteriormente descrevemos.

Na área assinalada com o número 2, temos um conjunto de botões que permitem invocar as operações do Web Service de selecção e manutenção de registos, para uma determinada base de dados e uma tabela seleccionada na área 1. Contém ainda um botão que permite aceder a um editor de instruções SQL, que serão enviadas ao wsQL.

Na área assinalada com o número 3, temos um conjunto de botões para invocar as operações que permitem obter do sistema wsQL o histórico de ligações, operações efectuadas e sucesso das mesmas, a chave pública e um botão que invoca a operação para fechar a ligação com o sistema wsQL.

Todos os dados retornados pelo wsQL são apresentados na área assinalada com o número 4.

5.7.3 Utilização da aplicação cliente

Feita a apresentação da aplicação cliente, chegou o momento de descrever a sua utilização no cenário apresentado.

Nesse sentido, e recorrendo ao écran da aplicação apresentado na figura 5.19, o utilizador define o URL do sistema wsQL a contactar, especifica o nome de utilizador e código de acesso ao sistema wsQL que está instalado na organização do URL definido. De seguida deve pressionar o botão *Login*, o qual executa o código apresentado no exemplo 5.5.

Exemplo 5.5 – Invocação das operações *GetChavePublica* e *Login*

```
1      Dim objwsQL As New wsQL.wsQL
2      Dim wsChavePublica As String
3
4      Private Sub btnLogin_Click
5
6          Try
7
8              objwsQL.Url = txtURL.Text
9              wsChavePublica = objwsQL.GetChavePublica
10
11          Catch ex As Exception
12
13              MessageBox.Show("Impossivel ligar ao Web service")
14              Exit Sub
15
16          End Try
17
18          Dim CodigoEncriptado As String
19          CodigoEncriptado = EncriptarRSA(txtCodigo.Text, wsChavePublica)
20
21          dsParametros = objwsQL.Login(txtNome.Text, CodigoEncriptado)
22
23      End Sub
```

Na linha 1 temos a criação de uma instância para o sistema wsQL. Na linha 8 é definido URL do sistema wsQL a contactar. Na linha 9 é invocada a primeira operação do wsQL que tem por objectivo obter a chave pública. Depois, a aplicação cliente deve encriptar o código de acesso com essa chave pública, (linha 19). Seguidamente, a aplicação cliente deve invocar a operação *Login* do wsQL com os respectivos parâmetros, como indicado na linha 21, a fim de aceder ao wsQL e proceder à sua validação perante a organização fornecedora do sistema wsQL. Por sua vez, o sistema wsQL, após validar a aplicação cliente, organiza a informação a retornar e envia-a à aplicação cliente. Essa informação constitui o documento XML do exemplo 5.4.

Depois da aplicação cliente processar essa informação, temos ao dispor o écran apresentado na figura 5.20.

Casos de utilização

O primeiro caso de utilização que apresentamos é o de requisitar uma tabela à organização fornecedora do sistema wsQL que está em Bragança usando a aplicação cliente que está em Gaia. Através da aplicação cliente, seleccionamos a base de dados e a tabela e pressionamos o botão *Dados tabela*, o qual invoca a operação correspondente no wsQL, com a sintaxe:

```
dsDados = objwsQL.GetDadosTabela(IDtemp, IDbd, Tabela)
```

Na Figura 5.21 temos o resultado da invocação da operação *GetDadosTabela* para a tabela “Alunos” da base de dados “Ligacao SGIS”.

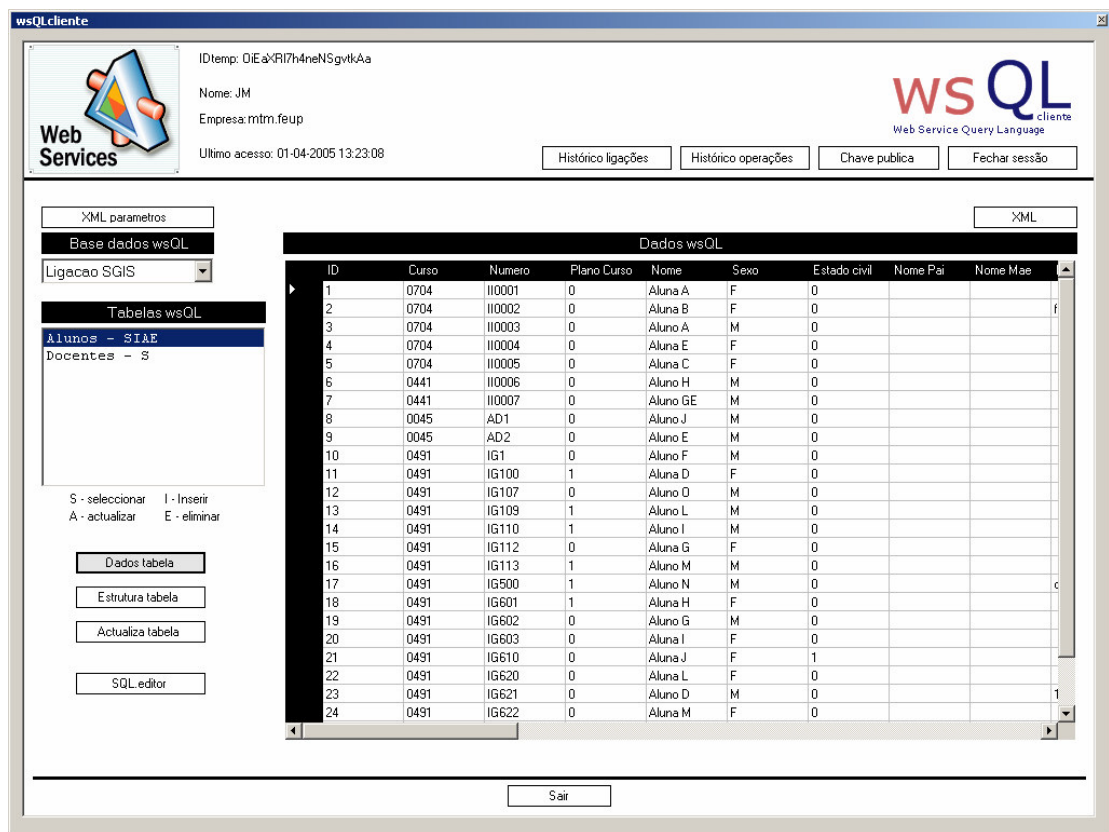


Figura 5.21 – Dados de uma tabela retornados pela operação *GetDadosTabela*

Contudo, a aplicação cliente nem sempre tem necessidade de receber todos os campos e registos de uma tabela. A capacidade fornecida pelo wsQL de autorizar a aplicação cliente a submeter instruções SQL permite, por um lado, filtrar a informação logo na origem, ou seja, no sistema wsQL e, por outro lado, reduzir a quantidade de informação a transferir entre o sistema wsQL e a aplicação cliente. Como a transferência é efectuada através de protocolos da Internet, essa redução de informação aumenta o desempenho geral do sistema.

O segundo caso de utilização consiste em apresentar a forma como a aplicação cliente pode submeter instruções SQL para serem processadas no sistema wsQL.

Para isso, no protótipo da aplicação cliente criámos um editor de instruções SQL, onde podem ser elaboradas as instruções de forma assistida. Depois de criada a instrução, a aplicação cliente submete-a ao sistema wsQL, utilizando a operação *ExecutaSQL*, com a seguinte sintaxe:

```
dsDados = objwsQL.ExecutaSQL(IDtemp, IDbd, txtSQL.Text)
```

Na figura 5.22 podemos observar o editor de instruções SQL, a instrução SQL elaborada e o resultado retornado após a submissão da mesma ao sistema wsQL.

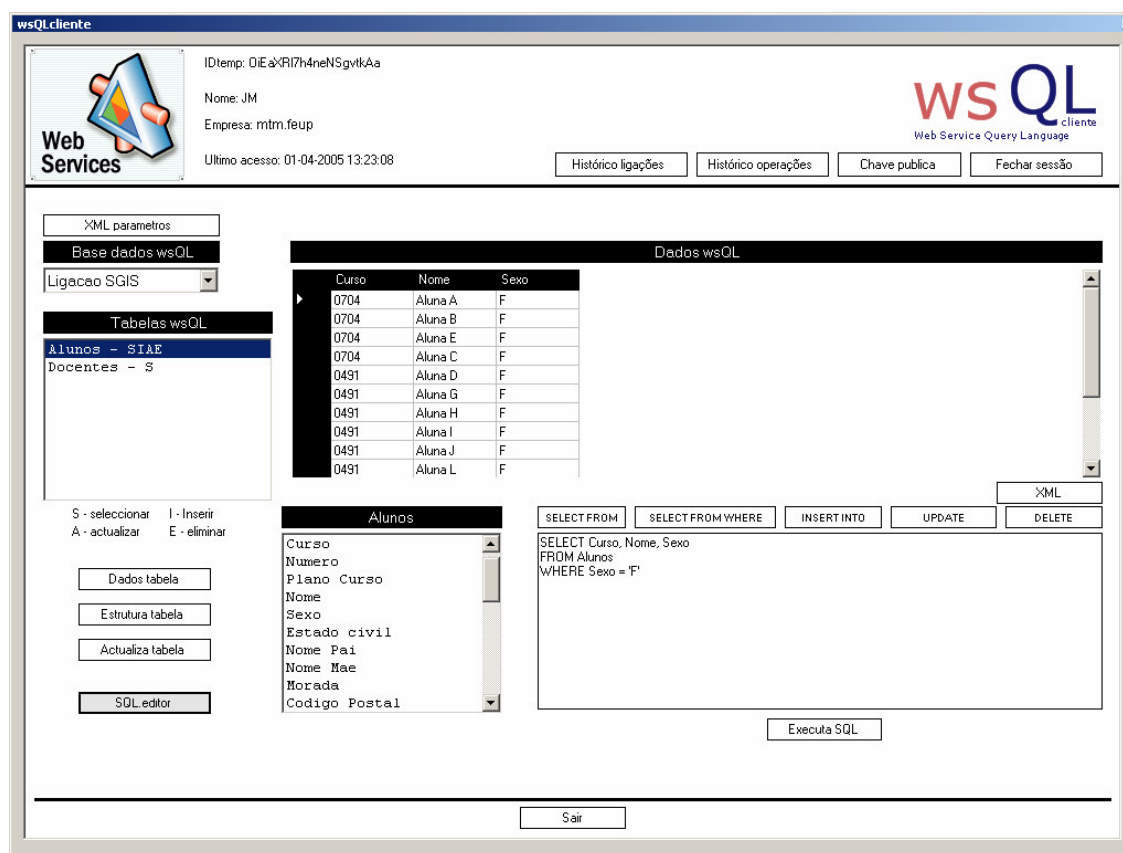


Figura 5.22 – Submeter instrução SQL ao sistema wsQL

5.8 Avaliação de conformidade do wsQL

Na secção anterior validamos o sistema wsQL num cenário real de integração de sistemas de informação escolares. Para isso, o sistema foi instalado nos primeiros quatro potenciais clientes. No entanto, uma vez que o sistema foi projectado e construído de forma a poder ser aplicado em qualquer contexto de integração de aplicações, torna-se necessário submeter o sistema a um processo de avaliação de conformidade segundo o perfil de interoperabilidade definido pela autoridade máxima de conformidade dos Web Services, o WS-I, *Web Services Interoperability Organization*.

O WS-I publica perfis de interoperabilidade, ferramentas de testes e aplicações exemplo para garantir que as aplicações construídas com as normas dos Web Services possam funcionar em conjunto [WS-I, 2005a] [WS-I, 2005b].

Na figura 5.23 podemos analisar o resultado, embora parcial, de conformidade do wsQL, produzido pela ferramenta de teste de interoperabilidade disponibilizada pelo WS-I. O Web Service wsQL passou no teste de conformidade do *Web Services Interoperability Organization*.

WS-I Profile Conformance Report



Report:	WS-I Basic Profile Conformance Draft Report. This is a prerelease version and no statement can be made from this report on WS-I conformance
Timestamp:	2005-05-02T22:03:43+01:00

Copyright (c) 2002-2003 by The Web Services-Interoperability Organization and Certain of its Members. All Rights Reserved.

Review the notice and license for information on the usage of this document. Also, feedback can be provided to the WS-I Organization.

Analyzer Tool Information

Version	1.0.1.7
Release Date	2004-01-20
Implementer Name	Web Services Interoperability Organization
Location	http://www.ws-i.org/implementation.aspx

Analyzer Runtime Environment Information

Runtime Name	.NET
Runtime Version	1.1.4322.2032
Operating System Name	Win32NT
Operating System Version	5.1.2600.0
XML Parser Name	.NET
XML Parser Version	1.1.4322.2032

Analyzer Configuration Options

Verbose	true
Type of Results to Show	all
Show Message Entry	true
Show Assertion Description	true
Show Failure Message	true
Show Failure Detail Message	true
Test Assertions Document	..\..\common\Profiles\BasicProfileTestAssertions.xml
Message Log File	..\samples\traceLog.xml
WSDL Element	
Name	wsQL
Type	binding
Namespace	http://fe.up.pt/wsQL/wsQL
Parent Element Name	[Not specified]
WSDL Document	http://localhost/wsQL/wsQL.asmx?wsdl
Service Location	http://localhost/wsQL/wsQL.asmx

Summary

Result	passed
---------------	---------------

Figura 5.23 - Resultado do teste de conformidade

Conclusão

É inegável que vivemos num tempo que certamente ficará registada nos anais da história como uma época de globalização crescente, sentida em todas as áreas da vida, tendo a Internet como principal motor impulsionador.

Como actualmente praticamente todas as organizações possuem sistemas de informação mais ou menos complexos, a integração e a interacção automática de aplicações torna-se vital para a competitividade e a evolução das organizações. No entanto, ao longo dos anos, a componente de integração e interacção automática de aplicações foi deixada de lado, o que resultou na criação de ilhas de dados no grande oceano dos sistemas de informação. Este problema acontece porque a grande competição entre as empresas de tecnologia tem impedido que se desenvolvam normas de comunicação.

Actualmente, os sistemas de informação distribuídos vivem momentos de grande agitação, especialmente depois da publicação da recomendação do W3C em Janeiro de 2002 para uma nova tecnologia, os Web Services, que surgiram com promessas de derrubar todas as barreiras que tem impedido a integração de aplicações a nível global, através da Internet.

Este trabalho pretendeu dar uma contribuição para a investigação desta tecnologia emergente e recheada de potencialidades. O problema estudado neste trabalho pertence à classe mais complexa dos problemas de sistemas de informação distribuídos para integração e interacção automática de aplicações utilizando protocolos da Internet. A sua importância é bem visível quando as organizações precisam que duas aplicações *falem* entre si, quer seja internamente quer seja com outras aplicações externas.

Nesse sentido, este trabalho tinha essencialmente dois objectivos. O primeiro era estudar a arquitectura orientada a serviços e as tecnologias mais representativas deste

modelo, estabelecendo uma ponte para o estudo do tema principal deste trabalho, os Web Services.

O segundo objectivo era a construção de uma aplicação seguindo o modelo de arquitectura de software orientada a serviços, utilizando o expoente máximo deste modelo, os Web Services, para interacção de aplicações recorrendo à teia mundial de comunicações que Internet constitui.

6.1 Trabalho desenvolvido

Este trabalho surgiu num contexto de agitação ao nível dos sistemas de informação distribuídos, no qual uma nova tecnologia associada a um novo modelo de programação tenta impor-se e afirmar-se.

O trabalho desenvolvido pode ser dividido em duas partes: numa primeira parte, apresentámos os conceitos e características por detrás da arquitectura orientada a serviços. Seguidamente, analisámos comparativamente as tecnologias mais representativas e amplamente usadas, evidenciando as principais limitações para compreender o que provocou a necessidade de definição de uma nova arquitectura conhecida como Web Services. A primeira parte continuou com o estudo desta arquitectura, onde apresentámos os fundamentos teóricos que a suportam, estabelecemos a relação com a arquitectura orientada a serviços, prosseguimos com o estudo dos protocolos que sustentam a arquitectura dos Web Services, agrupados em protocolos básicos e protocolos avançados e finalizámos esta primeira parte com as principais linhas de orientação para construir aplicações baseadas em Web Services.

A segunda parte do trabalho tinha por objectivo a construção de uma aplicação segundo o modelo da arquitectura de software orientada a serviços, com o propósito principal de demonstrar as capacidades deste novo paradigma constituído pelos Web Services. Para isso seria utilizado um problema simulado, com uma organização tipo com um determinado sistema de informação.

Depois de concluída a primeira parte, passámos para a actividade de construção da aplicação baseada em Web Services. Assim, após a realização do estudo inicial, reconhecemos que podíamos ser mais ambiciosos e desenvolver uma aplicação de âmbito geral, passível de ser utilizada em qualquer organização e para qualquer sistema de

informação, num contexto distribuído, utilizando a teia mundial da Internet para suportar a comunicação entre as aplicações.

Portanto, a nossa proposta centrou-se na elaboração de uma aplicação que consistiu numa nova camada de *software*, a que demos o nome wsQL – Web Services Query Language. A aplicação, ou seja a nova camada de *software*, é colocada entre a aplicação cliente, no requisitante dos serviços, e as bases de dados, no fornecedor dos serviços. O objectivo principal deste Web Service é permitir a uma organização expor, via *Web*, um conjunto de funções, para serem acedidas programaticamente, que recebam comandos SQL para interrogação ou manutenção de uma base de dados.

O sistema wsQL é constituído pela aplicação de gestão e pelo Web Service propriamente dito. Uma vez que o sistema foi projectado com um propósito de uso geral, toda a análise de requisitos foi efectuada no sentido de contemplar, por um lado, as funcionalidades que o gestor da aplicação de uma organização fornecedora dos serviços típica necessita para abrir o seu sistema de informação aos parceiros internos e externos e, por outro lado, as funcionalidades da aplicação cliente para usar as operações da aplicação wsQL.

Encontradas as fronteiras do sistema e identificados os requisitos funcionais e não funcionais, definimos de seguida a arquitectura modular do sistema wsQL. O sistema foi subdividido em três subsistemas principais: subsistema do Web Service, o subsistema de acesso aos sistemas de gestão de bases de dados, e o subsistema de gestão da aplicação.

Efectuada a análise, procedeu-se à implementação do sistema na plataforma .NET. A construção de cada subsistema foi orientada por uma abordagem de programação de componentes autónomos, adoptando o paradigma de desenvolvimento de aplicações seguindo o modelo de três camadas.

O sistema wsQL, através da validação a que foi submetido, num cenário real de integração de aplicações, demonstrou em pleno o seu potencial, ficando também evidenciadas as capacidades para ser utilizado noutros contextos.

Outro indicador de sucesso do sistema wsQL foi obtido através da submissão do Web Service ao teste de interoperabilidade do WS-I, que passou, uma vez que cumpriu os requisitos do perfil de interoperabilidade da ferramenta de *software* de teste.

6.2 Trabalho futuro

Ao longo da leitura desta dissertação, apercebemo-nos gradualmente de que o trabalho está inserido numa área em permanente ebulição e que novas normas e metodologias surgem a cada dia que passa, o que pode facilmente ser comprovado pela análise da tabela 3.3 ou então com uma visita ao sítio *Web* das organizações W3C e OASIS.

Apesar dos bons resultados obtidos, tanto na validação do sistema wsQL num cenário real, como na aprovação no teste de interoperabilidade do WS-I, e de resto como acontece com qualquer aplicação, o sistema wsQL pode ser dotado de novas funcionalidades de forma a ampliar as suas potencialidades.

Uma vez que a aplicação foi projectada segundo o perfil básico de interoperabilidade publicado pelo WS-I e implementada na plataforma .NET que foi projectada desde o início para inclui suporte aos Web Services, a curto prazo a manutenção do sistema consistirá principalmente em o adaptar às novas versões da plataforma de implementação e às novas versões das normas publicadas como recomendação.

A médio prazo, apontamos alguns novos requisitos agrupados em funcionais e não funcionais.

Requisitos funcionais

- Incluir no sistema wsQL o acesso a novos sistemas de gestão de bases de dados;
- Possibilitar à aplicação cliente executar no fornecedor dos serviços, através do wsQL, *Stored Procedures* de uma base de dados;
- Possibilitar a utilização a base de dados de imagens;
- Possibilidade de aceder a outros tipos de documentos.

Requisitos não funcionais

- Numa primeira fase, garantir a autenticação do cliente e da mensagem através da implementação do protocolo SSL ao nível de transporte;
- Numa segunda fase, implementar segurança total ao nível do SOAP, através da pilha de especificações do *WS-Security*;
- Numa terceira fase, a implementação dos protocolos para assegurar o encaminhamento de mensagens, a gestão de transacções e garantir a qualidade de serviço.

6.3 Considerações finais

Com a crescente evolução e complexidade dos sistemas de informação, especialmente, os que usam a Internet como espinha dorsal, novos requisitos surgiram, que impulsionaram a definição de um método que agregasse tais requisitos.

Segundo um estudo promovido pelo Aberdeen Group [Aberdeen, 2004] a cerca de 2000 responsáveis de sistemas de informação, estes apontaram o *e-Business* e o EAI como sendo as duas iniciativas mais importantes para a presente década.

O esforço para a definição de uma arquitectura que congregasse todos estes factores começou a partir do ano 2002 sobre a égide do W3C. Esta nova arquitectura apresentada como Web Services surgiu com grandes promessas de interoperabilidade nunca antes alcançada e de revolucionar todo o contexto das comunicações entre aplicações na *Web*. Eric Newcomer [Newcomer, 2002], faz mesmo a seguinte comparação: tal como a rede de transportes tem efeito na economia nacional de um país, os Web Services estão a promover fundamentais alterações nas regras do comércio via *Web*. Os Web Services ligam programas uns aos outros atravessando pontos distantes no mapa global, transportando grande volume de dados de uma forma mais eficiente e barata já alguma vez conseguida. Como resultado temos mais rapidez e maior produtividade nas comunicações para empresas e consumidores.

Mais uma vez na história da computação, as principais empresas de tecnologia informática estão associadas em diversos consórcios para estabelecer normas conjuntas, deixando a competição feroz entre elas para outros planos, por exemplo, ao nível de plataformas de desenvolvimento. Hoje, o mundo da programação de sistemas de informação divide-se em duas partes, a comunidade Java e a comunidade .NET.

A arquitectura dos Web Services apresenta características de tal forma potenciais, que rapidamente se vai impor como a principal tecnologia de integração de aplicações e de programação de sistemas de informação distribuídos em detrimento de tecnologias com um grande legado como CORBA, DCOM e JavaRMI.

Um estudo da Forrester Research, prevê que já neste ano de 2005 um grande volume de serviços seja gerado através da integração de aplicações com Web Services, estimando-se que em 2006 os Web Services sejam utilizados como base para novos modelos de negócios [Forrester Research, 2004].

Uma vez que a arquitectura protocolar básica dos Web Services recorre a normas largamente utilizadas na Internet, podemos dizer que se encontra definida e estabilizada;

no entanto, a arquitectura protocolar avançada que contempla aspectos como a segurança, encaminhamento de mensagens e a gestão de transacções, ainda estão a ser normalizados em alguns pontos. A nossa investigação teve início em Outubro 2003 e, nas sucessivas visitas aos sítios da *Web* do W3C e OASIS até Abril de 2005, observámos a publicação de novas normas e versões com grande regularidade.

A aplicação wsQL, surgiu num primeiro momento, para cumprir um requisito do curso de mestrado; no entanto, após a obtenção de bons resultados na validação, rapidamente nos apercebemos que tem potencial para se tornar numa aplicação comercial. Isto acontece porque a aplicação wsQL não foi desenvolvida para integrar um sistema de informação em particular; pelo contrário, foi construída com características de âmbito geral e assim foi dotada de potencialidades que permitem a sua utilização em qualquer contexto de partilha de base de dados. Claro que, para tornar a aplicação wsQL comercial, devem ser implementados os requisitos apontados na secção anterior.

Referências

- ABERDEEN** Group [2004], *Worldwide Enterprise Business Integration Spending: Forecast and Analysis 2002–2006*, Dezembro 2004, <http://www.aberdeen.com/default.asp>
- BERNERS-LEE** Tim [2002], *A Roadmap to the Semantic Web, and other whitepapers*, Novembro 2003, <http://www.w3.org/designissues/>
- CHAPPELL**, David [2005], *New Technologies Help You Make Your Web Services More Secure*, Janeiro 2005, <http://www.davidchappell.com/articles/>
- DOUGLAS**, K. Barry [2003], *Web Services and Service-Oriented Architecture: The Savvy Manager's Guide*, Morgan Kaufmann Publishers, 2003
- FORRESTER** Research [2004], *Trends 2005: Service-Oriented Architecture And Web Services*, 2004, <http://www.forrester.com/Research/Document/Excerpt/0,7211,35647,00.html>
- HANSON**, J. Jeffrey [2003], *.NET versus J2EE Web Services: A Comparison of Approaches*, Amazon.com, 2003
- IBM, MICROSOFT** [2005], *Security in a Web Services World: A Proposed Architecture and Roadmap*, A Joint White Paper from IBM Corporation and Microsoft Corporation, Janeiro 2005, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/securitywhitepaper.asp>
- IETF**, [2000a], *HTTP State Management Mechanism*, The Internet Society, 2000, <ftp://ftp.rfc-editor.org/in-notes/rfc2965.txt>
- IETF**, [2000b], *Upgrading to TLS Within HTTP/1.1*, The Internet Society, 2000, <http://www.apps.ietf.org/rfc/rfc2817.html>
- LI**, F. [1995], *The Geography of Business Information*, Wiley, 1995
- MICROSOFT** [2002a], *Building Secure ASP.NET Applications - Authentication, Authorization, and Secure Communication*, Microsoft Corporation, 2002
- MICROSOFT** [2005], *Microsoft Magazine*, Edição 50 > Janeiro-Março, Microsoft Portugal, 2005
- MSDN** [2005], *Web Services Developer Center*, 2005, <http://msdn.microsoft.com/webservices/>
- MSI** [2004], *Missão para a Sociedade da Informação*, “Livro Verde para a Sociedade da Informação em Portugal”, Janeiro 2004, <http://www.acesso.unic.pcm.gov.pt/docs/lverde.htm>

- NEWCOMER**, Eric [2002], *Understanding Web Services - XML, WSDL, SOAP and UDDI*, David Chappell series editor, 2002
- OASIS** [2005a], *eXtensible Access Control Markup Language TC v2.0 (XACML)*, 2005, <http://www.oasis-open.org/specs/index.php#xacmlv2.0>
- OASIS** [2005b], *Security Assertion Markup Language (SAML) v2.0*, 2005, <http://www.oasis-open.org/specs/index.php#samlv2.0>
- POTTS**, Stephen, **KOPACK**, Mike [2002], *Teach Yourself Web Services in 24 Hours*, Sams, 2002
- POTTS**, Stephen, **KOPACK**, Mike [2003], *Aprenda em 24 horas Web Services*, Editora Campos, 2003
- ROQUE**, V., **OLIVEIRA**, J. [2003], *CORBA, DCOM E JavaRMI – Uma Análise Comparativa*, IPG / IETA, 2003
- RSA Security** [2002], *RSA Algorithm*, <http://www.rsasecurity.com/rsalabs/node.asp?id=2146>
- SANTESSON**, S. et Al [2003], *Internet X.509 Public Key Infrastructure Qualified Certificates Profile*, ITU, 2003, <http://www.itu.int/rec/recommendation.asp?type=items&lang=E&parent=T-REC-X.509-200003-I>
- SLEEPER**, Brent [2001], “*Why UDDI Will Succeed, Quietly: Two Factors Push Web Services Forward*”, Stencil Group, 2001
- TOPLEY**, Kim [2003], *Java Web Services in Nutshell*, O'Reilly, 2003
- UDDI** [2004], *UDDI Version 3.0.2*, 2004, <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>
- W3C** [1996], World Wide Web Consortium, *Hypertext Transfer Protocol - HTTP/1.0*, The Internet Society, 1996 , <http://www.w3.org/Protocols/rfc1945/rfc1945>
- W3C** [1999], World Wide Web Consortium, *Hypertext Transfer Protocol - HTTP/1.1*, The Internet Society, 1999 , <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- W3C** [2005a], World Wide Web Consortium, *eXtensible Markup Language*, 2005, <http://www.w3.org/XML/>
- W3C** [2005b], World Wide Web Consortium, *Web Services Architecture*, 2005, <http://www.w3.org/TR/ws-arch/>
- W3C** [2005c], World Wide Web Consortium, *Web Services Description*, 2005, <http://www.w3.org/2002/ws/desc/>
- W3C** [2005d], World Wide Web Consortium, *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, 2005, <http://www.w3.org/TR/2005/WD-wsdl20-primer-20050510>
- W3C** [2005e], World Wide Web Consortium, *Web Services Glossary*, 2005, <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>

- W3C** [2005f], World Wide Web Consortium, *XML Encryption WG*, 2005, <http://www.w3.org/Encryption/2001/>
- W3C** [2005g], World Wide Web Consortium, *SOAP Version 1.2 Part 1: Messaging Framework*, 2005, <http://www.w3.org/TR/soap12-part1/>
- W3C** [2005h], World Wide Web Consortium, *XML Key Management*, 2005, <http://www.w3.org/2001/XKMS/>
- W3C** [2005i], World Wide Web Consortium, *XML Signature WG*, 2005, <http://www.w3.org/Signature/>
- WS-I** [2005a], *Basic Security Profile Version 1.0*, 2005, <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2005-01-20.html>
- WS-I** [2005b], *Security Challenges, Threats and Countermeasures*, 2005, <http://www.ws-i.org/Profiles/BasicSecurity/SecurityChallenges-1.00-ED-22.pdf>

Bibliografia

- AHMED**, Mesbah, **GARRETT**, Chris, **FAIRCLOTH**, Jeremy, **PAYNE**, Chris [2002], *ASP.NET Web Developer's Guide*, Syngress, 2002
- ANDERSON**, Steve et al. [2005a], *Web Services Secure Conversation Language (WS-Secure Conversation)*, BEA, IBM, Microsoft, RSA, etc., 2005
- ANDERSON**, Steve et al. [2005b], *Web Services Trust Language (WS-Trust)*, BEA, IBM, Microsoft, RSA, etc., 2005
- APTE**, Naresh, **MEHTA**, Toral [2002], *UDDI: Building Registry-Based Web Services Solutions*, Prentice Hall PTR, 2002
- ARMSTRONG**, Eric, Stephanie, Debbie, Maydene, Dale, Kim [2002], *The Java™ Web Services Tutorial*, Addison Wesley, 2002
- ATTIYA**, H., **WELCH**, J. [1998], *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw-Hill, 1998
- BALLINGER**, Keith [2003], *.NET Web Services: Architecture and Implementation*, Addison Wesley, 2003
- BAJAJ**, Siddharth et al. [2004], *Web Services Federation Language (WS-Federation)*, BEA, IBM, Microsoft, RSA, VeriSign, 2004
- BANERJEE**, Ashish, Aravind, Zach, Andrew, Christian, Chris, Thiru, Brad [2003], *C# Web Services - Building Web Services with .NET Remoting and ASP.NET*, Wrox, 2003
- BEA** [2001], *Developing Web Services on the J2EE Platform*, BEA Systems, Inc., 2001
- BEVINAKOPPA**, Savitri, Hu, Jiankun [2004], *Web Services: Modeling, Architecture and Infrastructure, WSMAl 2004 - ICEIS*, Insticc, 2004
- BOAR**, C. [2003], *XML Web Services in the Organization*, Microsoft Press, 2003
- BORLAND** [2002], *Web Services Development Tools Compared*, Borland Software Corporation, 2002
- CABRERA**, L. F. et al. [2004a], *Web Services Atomic Transaction (WS-AtomicTransaction)*, BEA, IBM, Microsoft, 2004
- CABRERA**, L. F. et al. [2004b], *Web Services Coordination (WS-Coordination)*, BEA, IBM, Microsoft, 2004
- CAPECLEAR** [2004], *Principles of SOA Design*, Capeclear, 2004

- CASTRO-LEON**, Enrique [2002], *A perspective on Web Services*, 2002, <http://www.webservices.org>
- CERAMI**, Ethan [2002], *Web Services Essentials*, O'Reilly, 2002
- CHAPPELL**, David e **JWELL**, Tyler [2002], *Java Web Services*, O'Reilly, 2002
- CLABBY**, Joe [2002], *Web Services Explained: Solutions and Applications for the Real World*, Prentice Hall PTR, 2002
- COYLE**, Frank P. [2002], *XML, Web Services, and the Data Revolution*, Addison Wesley, 2002
- CUNNINGHAM**, W., *Enterprise Solution Patterns Using Microsoft .NET*, Microsoft Press, 2003
- CURTIS**, D. [1997], *Java, RMI and CORBA*, OMG, 1997
- DAMAS**, Luís [2000], *SQL*, FCA, 2000
- DOBSON**, Rick [2002], *Programming Microsoft® SQL Server™ 2000 with Microsoft Visual Basic® .NET*, Microsoft Corporation, 2002
- GUNDERLOY**, Mike [2003], *MCAD/MCSD Training Guide (70-310): Developing XML Web Services and Server Components with Microsoft® Visual Basic® .NET and the Microsoft .NET Framework*, Que, 2003
- GURUGÉ** Anura [2003], *Corporate Portals Empowered with XML and Web Services*, Digital Press, 2003
- GRONMO**, Roy [2004], *Towards Modeling Web Services Composition in UML*, Sintef, 2004
- HAROLD**, Elliotte Rusty, **MEANS**, W. Scott Means [2002], *XML in a Nutshell, 2nd Edition*, O'Reilly, 2002
- HARTMAN** Bret, **FLINN**, Donald J., **BEZNOSOV**, Konstantin, **KAWAMOTO**, Shirley, [2003] *Mastering Web Services Security*, Wiley, 2003
- HOCHGURTEL**, Brian [2003], *Cross-Platform Web Services Using C# and Java*, Charles River Media, 2003
- IBM** [2004a], *Patterns: Service-Oriented Architecture and Web Services*, IBM, Maio 2004, <http://www.redbooks.ibm.com/>
- IBM** [2004b], *Using Web Services for services Business Integration*, IBM, Maio 2004, <http://www.redbooks.ibm.com/>
- IBM** [2004c], *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, Maio 2004, <http://www.redbooks.ibm.com/>
- IBM** [2004d], *WebSphere Web Services Information Roadmap*, IBM, Maio 2004, <http://www.redbooks.ibm.com/>

- IDC** [2004], *Web Services are Becoming Reality*, <http://www.idc.com>, 2004
- IONA** [2004], *Web Services Foundations*, IONA Technologies PLC, 2004
- JAMSA** [2003], K., *.NET Web Services Solutions*, Sybex, 2003
- JORGENSEN**, David [2002], *Developing .NET Web Services With XML*, Syngress Publishing, Inc., 2002
- KAYE**, Doug [2003], *Loosely Coupled: The Missing Pieces of Web Services*, RDS Press, 2003
- KIMMEL**, Paul [2003], *Visual Basic® .NET Power Coding*, Addison Wesley, 2003
- KINSMAN**, Chris, Mcmanus, Jeffrey [2001], *VB.NET Developer's Guide to ASP.NET, XML, and ADO.NET*, Sams, 2001
- KULCHENKO**, Pavel, RAY Randy J. [2002], *Programming Web Services with Perl*, O'Reilly, 2002
- KURNIAWAN**, Budi, NEWARD, Ted [2002], *VB.NET Core Classes in a Nutshell*, O'Reilly, 2002
- LOMAX**, Paul, PETRUSHA, Ron, ROMAN, Steven [2002a], *VB.NET Language in a Nutshell, 2nd Edition*, O'Reilly, 2002
- LOMAX**, Paul, PETRUSHA, Ron, ROMAN, Steven [2002b], *VB.NET Language Pocket Reference*, O'Reilly, 2002
- MACDONALD**, Matthew [2003], *Microsoft® .NET Distributed Applications: Integrating XML Web Services and .NET Remoting*, Microsoft Press, 2003
- MANES**, Anne Thomas [2003], *Web Services: A Manager's Guide*, Addison Wesley, 2003
- MARCHAL**, Benoît [2000], *XML by Example*, Que, 2000
- MARQUES**, JP., GUEDES, P. [1998], *Tecnologia de Sistemas Distribuídos*, FCA, 1998
- MARTINS**, Ricardo, ROCHA, Jorge, HENRIQUES, Pedro [2003], *Segurança dos Web Services no Comércio Eletrónico Móvel*, Universidade do Minho, 2003
- MCGOVERN**, James, Sameer, Michael e Sunil [2003], *Java Web Services Architecture*, Morgan Kaufmann Publishers, 2003
- META** Group Consulting [1998], *CORBA vs. DCOM: Solutions for the Enterprise*, 1998
- MICROSOFT** [2002b], *Application Architecture for .NET: Designing Applications and Services*, Microsoft Corporation, 2002
- MICROSOFT** [2002c], *Building XML Web Services For The Microsoft .NET Platform*, Microsoft Corporation, 2002

- MICROSOFT** [2002d], *Global XML Web Services Architecture*, Microsoft Corporation, 2002
- MICROSOFT** [2003], *MCAD/MCSD Training Kit Developing XML Web Services and Server Components with Microsoft Visual Basic .NET and Microsoft Visual C# .NET*, Microsoft Corporation, 2003
- MICROSOFT** [2004], *Building Interoperable Web Services: WS-I Basic Profile*, Microsoft Corporation, 2004
- MITTAL**, Kunal [2003], *Web Services and the Real Estate Industry*, Tect, 2003
- MONSON-HAEFEL**, Richard [2003], *J2EE™ Web Services*, Addison Wesley, 2003
- MOREIRA**, José J. [2003], *Plataforma .NET e Web Services*, FEUP, 2003
- MSDN** [2002], *Developing XML Web Services Using Microsoft® ASP.NET*, Microsoft, 2002
- MUKHI**, Vijay, **RANCHAND**, Vikram, **MUKHI**, Sonal [2002], *XML Web Services and SOAP*, BPB Publications, 2002
- NGHIEM** [2002], Alex, *IT Web Services: A Roadmap for the Enterprise*, Prentice Hall PTR, 2002
- OASIS** [2004], *WS-Reliability v1.1*, 2004, <http://www.oasis-open.org/specs/index.php#wsrv1.1>
- OMG** [2004], *Java, RMI and CORBA*, 2004, <http://www.omg.org/library/wpjava.html>
- OMG** [2004], *The Common Object Request Broker: Architecture and Specification*, Março 2004, http://www.omg.org/technology/documents/corba_spec_catalog.htm
- O'NEIL**, Mark et al. [2003], *Web Service Security*, McGraw-Hill/Osborne, 2003
- PARDI**, William J. [1999], *XML in action*, Microsoft Press, 1999
- PARIHAR**, Mridula [2002], *ASP.NET Bible*, Hungry Minds, 2002
- PROVOST**, Will [2004], *UML for Web Services*, XML.com, 2004
- RAY**, Erik T. [2001], *Learning XML*, O'Reilly, 2001
- RYAN**, D. e **RYAN T.** [2003], *Your visual Blueprint for Creating Web Applications on the .NET framework*, MaranGraphics & Hungry Minds, 2003
- SCRIBNER**, Kenn, **STIVER**, Mark C. [2001], *Applied SOAP: Implementing .NET XML Web Services*, Sams Publishing, 2001
- SILVA**, Miguel M. [2003], *Integração de Sistemas de Informação*, FCA, 2003
- SKONNARD**, Aaron, **GUDGIN**, Martin [2002], *Essential XML Quick Reference*, Addison Wesley, 2002

- SPROTT D.** e **WILKES L.** [2004], *Understanding Service-Oriented Architecture*, CBDI Forum, 2004
- SNELL**, James [2001], *Programming Web Services with SOAP*, O'Reilly, 2001
- ST. LAURENT**, Simon, **JOHNSTON**, Joe, **DUMBILL**, Edd [2001], *Programming Web Services with XML-RPC*, O'Reilly, 2001
- STURM**, Jake [2000], *Developing XML Solutions*, Microsoft Press, 2000
- SUN** [2001], *Web Services Made Easier*, Sun Microsystems, Inc. 2001
- SYBEX** [2002], *VB.NET Functions and Statements*, Sybex, 2002
- SYSTINET** [2003], *Web Services: A Practical Introduction*, 2003, <http://www.systinet.com>
- TAUBER**, James [2003], *The Web Services Revolution*, Sun Microsystems, Inc. 2003
- TOFFLER**, A. [1984], *A Terceira Vaga*, Livros do Brasil, 1984.
- TRAVIS**, Brian E. [2000], *XML and SOAP Programming for BizTalk Servers*, Microsoft Corporation, 2000
- VARAJÃO**, J. E. Q. [1998], *A Arquitetura da Gestão de Sistemas de Informação*, FCA, 1998
- VINOSKI**, Steve [2004], *Extending Web Services Bindings for Real-world Enterprise Computing Systems*, IONA, 2004, <http://www.iona.com/hyplan/vinoski>
- WEBMETHODS** [2004], *Demystifying Web Services*, <http://webmethods.com>, 2004
- WILLIAM**, L., **OELLERMANN**, Jr. [2001], *Architecting Web Services*, Apress, 2001
- WAINEWRIGHT**, Phil [2002], *Web Services Infrastructure - The global utility for real-time business*, Phil Wainewright, 2002
- W3C** [2005j], World Wide Web Consortium, *Web Services Architecture Requirements*, 2005, <http://www.w3.org/TR/wsa-reqs/>
- W3C** [2005l], World Wide Web Consortium, *Web Services Choreography*, 2005, <http://www.w3.org/2002/ws/chor/>

Anexos

Documento XML com esquema XML - XSD

Exemplo A.1 – Documento XML com esquema XML

```
<?xml version="1.0" encoding="utf-8"?>
<DataSet xmlns="http://fe.up.pt/wsQL/wsQL">
  <xs:schema id="NewDataSet" xmlns=""
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:Locale="pt-PT">
      <xs:complexType>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="Parametros">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="IDutilizador" type="xs:string" minOccurs="0" />
                <xs:element name="Nome" type="xs:string" minOccurs="0" />
                <xs:element name="Empresa" type="xs:string" minOccurs="0" />
                <xs:element name="UltimoAcesso" type="xs:dateTime" minOccurs="0" />
                <xs:element name="IDbd" type="xs:int" minOccurs="0" />
                <xs:element name="BaseDados" type="xs:string" minOccurs="0" />
                <xs:element name="Tabela" type="xs:string" minOccurs="0" />
                <xs:element name="Seleccionar" type="xs:boolean" minOccurs="0" />
                <xs:element name="Inserir" type="xs:boolean" minOccurs="0" />
                <xs:element name="Atualizar" type="xs:boolean" minOccurs="0" />
                <xs:element name="Apagar" type="xs:boolean" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">

    <NewDataSet xmlns="">
      <Parametros diffgr:id="Parametros1" msdata:rowOrder="0" diffgr:hasChanges="inserted">
        <IDutilizador>xhp3McDgILvGcVxd6SX7</IDutilizador>
        <Nome>JM</Nome>
        <Empresa>XPTO</Empresa>
        <UltimoAcesso>2004-11-14T18:49:00.0000000-00:00</UltimoAcesso>
      </Parametros>
      <Parametros diffgr:id="Parametros2" msdata:rowOrder="1" diffgr:hasChanges="inserted">
        <IDbd>1</IDbd>
        <BaseDados>Ligacao SISLA</BaseDados>
        <Tabela>Alunos</Tabela>
        <Seleccionar>true</Seleccionar>
        <Inserir>true</Inserir>
        <Atualizar>true</Atualizar>
        <Apagar>false</Apagar>
      </Parametros>
    </NewDataSet>
  </diffgram>
</DataSet>
```

```
</Parametros>
<Parametros diffgr:id="Parametros3" msdata:rowOrder="2" diffgr:hasChanges="inserted">
  <IDbd>1</IDbd>
  <BaseDados>Ligacao SISLA</BaseDados>
  <Tabela>Docentes</Tabela>
  <Seleccionar>true</Seleccionar>
  <Inserir>false</Inserir>
  <Atualizar>false</Atualizar>
  <Apagar>false</Apagar>
</Parametros>
<Parametros diffgr:id="Parametros4" msdata:rowOrder="3" diffgr:hasChanges="inserted">
  <IDbd>2</IDbd>
  <BaseDados>Ligacao SGIS</BaseDados>
  <Tabela>Alunos</Tabela>
  <Seleccionar>true</Seleccionar>
  <Inserir>false</Inserir>
  <Atualizar>false</Atualizar>
  <Apagar>false</Apagar>
</Parametros>
</NewDataSet>
</diffgr:diffgram>
</DataSet>
```

Modelo de dados da Aplicação de Gestão

Tabela B.1 – Tabela Utilizadores

UTILIZADORES			
	CAMPO	TIPO DE DADOS	COMPRIMENTO
	ID	INT	4
	UTILIZADOR	NVARCHAR	30
	CODIGO	NVARCHAR	10
	NOME	NVARCHAR	50
	EMPRESA	NVARCHAR	50
	ULTIMOACESSO	DATETIME	8

Tabela B.2 – Tabela PartilhasBds

PARTILHASBDS			
	CAMPO	TIPO DE DADOS	COMPRIMENTO
	ID	INT	4
	DESCRICAO	NVARCHAR	30
	TIPO	NVARCHAR	6
	CAMINHOMDB	NVARCHAR	50
	BdMDB	NVARCHAR	50
	CHAVEMDB	NVARCHAR	8
	SERVIDORSQL	NVARCHAR	50
	BdSQL	NVARCHAR	50
	AUTENTICACAOSQL	INT	4
	USERSQL	NVARCHAR	30
	CHAVESQL	NVARCHAR	15
	SERVIDORORA	NVARCHAR	50
	BdORA	NVARCHAR	50
	USERORA	NVARCHAR	30
	CHAVEORA	NVARCHAR	15

Tabela B.3 – Tabela UtilizadoresPartilhas

UTILIZADORESPARTILHAS			
	CAMPO	TIPO DE DADOS	COMPRIMENTO
	ID	INT	4
	IDBD	INT	4
	IDUTILIZADOR	INT	4
	TABELA	NVARCHAR	50
	SELECIONAR	BOOLEAN	1
	INSERIR	BOOLEAN	1
	ACTUALIZAR	BOOLEAN	1
	APAGAR	BOOLEAN	1

Tabela B.4 – Tabela LigacoesOnLine

LIGACOESONLINE			
	CAMPO	TIPO DE DADOS	COMPRIMENTO
	ID	INT	4
	IDUTILIZADOR	INT	4
	IDUTILIZADORTMP	NVARCHAR	50
	IDHISTORICO	INT	4

Tabela B.5 – Tabela LigacoesHistorico

LIGACOESHISTORICO			
	CAMPO	TIPO DE DADOS	COMPRIMENTO
	ID	INT	4
	IDUTILIZADOR	INT	4
	INICIO	DATETIME	8
	FIM	DATETIME	8

Tabela B.6 – Tabela HistoricoOperacoes

HISTORICOOPERACOES			
	CAMPO	TIPO DE DADOS	COMPRIMENTO
	ID	INT	4
	IDUTILIZADOR	INT	4
	DATA	DATETIME	8
	OPERACAO	NVARCHAR	50
	BASEDADOS	NVARCHAR	50
	TABELA	NVARCHAR	50
	SUCESSO	BOOLEAN	1

Código da Aplicação wsQL

C.1 Código do subsistema do Web Service

Exemplo C.1.1 – Código do Web Service

```
Imports System.Web.Services
Imports System.Security.Cryptography
<System.Web.Services.WebService(Namespace:="http://fe.up.pt/wsQL/wsQL")> _
Public Class wsQL
    Inherits System.Web.Services.WebService
    Dim objDadoswsQL As New AcessoDados.clsAcessoDadosSQL
    Private Shared InfoChavePublica As String
    Private Shared InfoChavePrivada As String

    <WebMethod(Description:="Pedir chave publica")> _
    Public Function GetChavePublica() As String
        If (InfoChavePrivada = Nothing) Then
            Dim CSPparam As CspParameters
            Dim RSA As RSACryptoServiceProvider
            CSPparam = New CspParameters
            CSPparam.Flags = CspProviderFlags.UseMachineKeyStore
            RSA = New RSACryptoServiceProvider(CSPparam)
            InfoChavePublica = RSA.ToXmlString(False)
            InfoChavePrivada = RSA.ToXmlString(True)
        End If
        Return InfoChavePublica
    End Function

    <WebMethod(Description:="Efetuar login com codigo encriptado")> _
    Public Function Login(ByVal Utilizador As String, _
        ByVal Codigo As String) As DataSet
        Dim dsParametros As DataSet
        Dim CodigoDesencriptado As String
        Dim ChavePublicaCliente As String = ""
        CodigoDesencriptado = DesencriptarRSA(Codigo, InfoChavePrivada)
        dsParametros = ValidaUtilizador(Utilizador, CodigoDesencriptado, ChavePublicaCliente)
        Return dsParametros
    End Function

    <WebMethod(Description:="Fechar sessão")> _
    Public Function FechaSessao(ByVal IDtemp As String) As Boolean
        Dim IDutilizador As Integer
        Dim IDhistorico As Integer
        Dim dsLigHist As DataSet
        Dim Sucesso As Boolean
        IDutilizador = BuscaIDutilizador(IDtemp)
        IDhistorico = BuscaIDhistorico(IDtemp)
        Dim strSQL As String
        strSQL = "SELECT * FROM LigacoesHistorico" & _
            " WHERE ID = " & IDhistorico
        dsLigHist = objDadoswsQL.CriaDataSet("LigacoesHistorico", strSQL)
```

```

With dsLigHist.Tables(0)
    If .Rows.Count > 0 Then
        .Rows(0)("Fim") = Now 'Regista data hora de fim no historico
    End If
End With
Sucesso = objDadoswsQL.UpdateDataSet(dsLigHist, "LigacoesHistorico")
Dim dsLigOnline As DataSet
strSQL = "SELECT *" & _
        " FROM LigacoesOnLine" & _
        " WHERE IDutilizador = " & IDutilizador
dsLigOnline = objDadoswsQL.CriaDataSet("LigacoesOnLine", strSQL)
With dsLigOnline.Tables(0)
    If .Rows.Count > 0 Then
        .Rows(0).Delete() 'Retira utilizador na tabela LigacoesOnline
    End If
End With
Sucesso = objDadoswsQL.UpdateDataSet(dsLigOnline, "LigacoesOnLine")
Return Sucesso
End Function

<WebMethod(Description:="Devolve a estrutura de uma tabela")> _
Public Function GetEstruturaTabela(ByVal IDtemp As String, _
                                   ByVal IDbd As Integer, _
                                   ByVal Tabela As String) As DataSet

    Dim IDutilizador As Integer
    Dim TipoLig As String
    Dim CaminhoMDB As String
    Dim BdMDB As String
    Dim ServidorSQL As String
    Dim bdSQL As String
    Dim UserIDSQL As String
    Dim PasswordSQL As String
    Dim AutenticacaoSQL As Integer
    Dim Conn As String
    Dim Caminho As String
    BuscaLigacao(IDtemp, IDbd, Tabela, IDutilizador, _
                TipoLig, ServidorSQL, bdSQL, AutenticacaoSQL, UserIDSQL, PasswordSQL, _
                CaminhoMDB, BdMDB, Conn, Caminho)
    Dim dsEstruturaTab As DataSet
    Dim strSQL As String
    Dim Sucesso As Boolean
    Select Case TipoLig
        Case "SQL"
            strSQL = "SELECT TOP 0 * FROM " & Tabela
            Dim objDadosSQL As New AcessoDados.clsAcessoDadosSQL(Conn)
            dsEstruturaTab = objDadosSQL.CriaDataSet(Tabela, strSQL)
        Case "MDB"
            strSQL = "SELECT top 1 * FROM " & Tabela
            Dim objDadosOLEDB As New AcessoDados.clsAcessoDadosOLEDB(Caminho)
            dsEstruturaTab = objDadosOLEDB.CriaDataSet(Tabela, strSQL)
    End Select
    If dsEstruturaTab Is Nothing Then
        Sucesso = False
    Else
        Sucesso = True
    End If
    GuardaOperacaoUtilizador(IDutilizador, "GetEstruturaTabela", IDbd, Tabela, Sucesso)
    Return dsEstruturaTab
End Function

<WebMethod(Description:="Devolve os registos de uma tabela")> _
Public Function GetDadosTabela(ByVal IDtemp As String, _
                               ByVal IDbd As Integer, _
                               ByVal Tabela As String) As DataSet

    Dim IDutilizador As Integer
    Dim TipoLig As String
    Dim CaminhoMDB As String
    Dim BdMDB As String
    Dim ServidorSQL As String
    Dim bdSQL As String
    Dim UserIDSQL As String
    Dim PasswordSQL As String
    Dim AutenticacaoSQL As Integer
    Dim Conn As String
    Dim Caminho As String

```

```

BuscaLigacao(IDtemp, IDbd, Tabela, IDutilizador, _
              TipoLig, ServidorSQL, bdSQL, AutenticacaoSQL, UserIDSQL, PasswordSQL, _
              CaminhoMDB, BdMDB, Conn, Caminho)
Dim dsDadosTab As DataSet
Dim Sucesso As Boolean

Select Case TipoLig
    Case "SQL"
        Dim objDadosSQL As New AcessoDados.clsAcessoDadosSQL(Conn)
        dsDadosTab = objDadosSQL.CriaDataSet(Tabela)
    Case "MDB"
        Dim objDadosOLEDB As New AcessoDados.clsAcessoDadosOLEDB(Caminho)
        dsDadosTab = objDadosOLEDB.CriaDataSet(Tabela)
End Select
If dsDadosTab Is Nothing Then
    Sucesso = False
Else
    Sucesso = True
End If
GuardaOperacaoUtilizador(IDutilizador, "GetDadosTabela", IDbd, Tabela, Sucesso)
Return dsDadosTab
End Function

<WebMethod(Description:="Executa comandos SQL numa base de dados")> _
Public Function ExecutaSQL(ByVal IDtemp As String, _
                          ByVal IDbd As Integer, _
                          ByVal strSQL As String) As DataSet
    Dim dsDadosTab As DataSet
    Dim Sucesso As Boolean
    Dim IDutilizador As Integer
    Dim TipoLig As String
    Dim CaminhoMDB As String
    Dim BdMDB As String
    Dim ServidorSQL As String
    Dim bdSQL As String
    Dim UserIDSQL As String
    Dim PasswordSQL As String
    Dim AutenticacaoSQL As Integer
    Dim Conn As String
    Dim Caminho As String
    BuscaLigacao(IDtemp, IDbd, IDutilizador, _
                  TipoLig, ServidorSQL, bdSQL, AutenticacaoSQL, UserIDSQL, PasswordSQL, _
                  CaminhoMDB, BdMDB, Conn, Caminho)
    Select Case TipoLig
        Case "SQL"
            Dim objDadosSQL As New AcessoDados.clsAcessoDadosSQL(Conn)
            dsDadosTab = objDadosSQL.CriaDataSet("Dados", strSQL)
        Case "MDB"
            Dim objDadosOLEDB As New AcessoDados.clsAcessoDadosOLEDB(Caminho)
            dsDadosTab = objDadosOLEDB.CriaDataSet("Dados", strSQL)
    End Select
    If dsDadosTab Is Nothing Then
        Sucesso = False
    Else
        Sucesso = True
    End If
    GuardaOperacaoUtilizador(IDutilizador, "ExecutaSQL", IDbd, strSQL, Sucesso)
    Return dsDadosTab
End Function

<WebMethod(Description:="Atualiza dados de uma tabela (Insert; Update; Delete)")> _
Public Function AtualizaDadosTabela(ByVal IDtemp As String, _
                                    ByVal IDbd As Integer, _
                                    ByVal TipoAtualizacao As Char, _
                                    ByVal Tabela As String, _
                                    ByVal Valores As String, _
                                    ByVal Campos As String) As Boolean
    Dim IDutilizador As Integer
    Dim TipoLig As String
    Dim CaminhoMDB As String
    Dim BdMDB As String
    Dim ServidorSQL As String
    Dim bdSQL As String
    Dim UserIDSQL As String
    Dim PasswordSQL As String
    Dim AutenticacaoSQL As Integer
    Dim Conn As String

```

```

Dim Caminho As String
BuscaLigacao(IDtemp, IDb, IDutilizador, _
    TipoLig, ServidorSQL, bdSQL, AutenticacaoSQL, UserIDSQL, PasswordSQL, _
    CaminhoMDB, BdMDB, Conn, Caminho)
Dim Sucesso As Boolean

Select Case TipoLig
    Case "SQL"
        Dim objDadosSQL As New AcessoDados.clsAcessoDadosSQL(Conn)
        Select Case TipoAtualizacao
            Case "I"
                Sucesso = objDadosSQL.ExecutaINSERT(Tabela, Campos, Valores)
            Case "U"
                Sucesso = objDadosSQL.ExecutaUPDATE(Tabela, Valores, Campos)
            Case "D"
                Sucesso = objDadosSQL.ExecutaDELETE(Tabela, Campos)
        End Select
    Case "MDB"
        Dim objDadosOLEDB As New AcessoDados.clsAcessoDadosOLEDB(Caminho)
        Select Case TipoAtualizacao
            Case "I"
                Sucesso = objDadosOLEDB.ExecutaINSERT(Tabela, Valores, Campos)
            Case "U"
                Sucesso = objDadosOLEDB.ExecutaUPDATE(Tabela, Valores, Campos)
            Case "D"
                Sucesso = objDadosOLEDB.ExecutaDELETE(Tabela, Campos)
        End Select
    End Select
GuardaOperacaoUtilizador(IDutilizador, "Atualiza Tabela", IDb, "", Sucesso)
Return Sucesso
End Function

<WebMethod(Description:="Devolve histórico de ligacoes do cliente")> _
Public Function HistoricoLigacoes(ByVal IDtemp As String) As DataSet
    Dim IDutilizador As Integer
    Dim dsLigHist As DataSet
    Dim Sucesso As Boolean
    IDutilizador = BuscaIDutilizador(IDtemp)
    Dim strSQL As String
    strSQL = "SELECT Inicio, Fim" & _
        " FROM LigacoesHistorico" & _
        " WHERE IDutilizador = " & IDutilizador & _
        " ORDER BY Inicio DESC"
    dsLigHist = objDadoswsQL.CriaDataSet("LigacoesHistorico", strSQL)
    If dsLigHist Is Nothing Then
        Sucesso = False
    Else
        Sucesso = True
    End If
    GuardaOperacaoUtilizador(IDutilizador, "HistoricoLigacoes", "", "", Sucesso)
    Return dsLigHist
End Function

<WebMethod(Description:="Devolve histórico de operações efectuadas")> _
Public Function HistoricoOperacoes(ByVal IDtemp As String) As DataSet
    Dim IDutilizador As Integer
    Dim dsLigHist As DataSet
    Dim Sucesso As Boolean
    IDutilizador = BuscaIDutilizador(IDtemp)
    Dim strSQL As String
    strSQL = "SELECT Data, Operacao, Tabela, Sucesso" & _
        " FROM HistoricoOperacoes" & _
        " WHERE IDutilizador = " & IDutilizador & _
        " ORDER BY Data DESC"
    dsLigHist = objDadoswsQL.CriaDataSet("HistoricoOperacoes", strSQL)
    If dsLigHist Is Nothing Then
        Sucesso = False
    Else
        Sucesso = True
    End If
    GuardaOperacaoUtilizador(IDutilizador, "HistoricoOperacoes", "", "", Sucesso)
    Return dsLigHist
End Function
End Class

```

C.2 Componente Modulo

Exemplo C.2.1 – Código do Modulo

```
Imports System.Security.Cryptography
Module Modulo
    Dim objDadoswsQL As New AcessoDados.clsAcessoDadosSQL
    Public dsUtilizadores As DataSet

    ' Funções internas
    Function EncriptarRSA(ByVal TextoParaEncriptar As String, _
        ByVal ChavePublica As String) As String
        Dim ParaEncriptar As Byte()
        Dim TextoEncriptado As Byte()
        Dim RSA As RSACryptoServiceProvider
        ParaEncriptar = System.Text.UnicodeEncoding.Unicode.GetBytes(TextoParaEncriptar)
        RSA = New RSACryptoServiceProvider
        RSA.FromXmlString(ChavePublica)
        TextoEncriptado = RSA.Encrypt(ParaEncriptar, False)
        Return System.Convert.ToBase64String(TextoEncriptado)
    End Function

    Function DesencriptarRSA(ByVal TextoParaDesencriptar As String, _
        ByVal ChavePrivada As String) As String
        Dim CSPparam As CspParameters
        Dim RSA As RSACryptoServiceProvider
        Dim BytesParaDesencriptar As Byte()
        Dim TextoDesencriptado As Byte()
        CSPparam = New CspParameters
        CSPparam.Flags = CspProviderFlags.UseMachineKeyStore
        RSA = New RSACryptoServiceProvider(CSPparam)
        RSA.FromXmlString(ChavePrivada)
        BytesParaDesencriptar = System.Convert.FromBase64String(TextoParaDesencriptar)
        TextoDesencriptado = RSA.Decrypt(BytesParaDesencriptar, False)
        Return System.Text.UnicodeEncoding.Unicode.GetString(TextoDesencriptado)
    End Function

    Function ValidaUtilizador(ByVal Utilizador As String, _
        ByVal Codigo As String, _
        ByVal ChavePublicaCliente As String) As DataSet
        Dim dsParametros As DataSet
        Dim dsTabelas As DataSet
        Dim dRowP As DataRow
        If UCase(Utilizador) = "DEMO" And UCase(Codigo) = "DEMO" Then
            dsParametros = CriaDataSet()
            dRowP = dsParametros.Tables(0).NewRow()
            dRowP("IDutilizador") = "DEMO"
            dRowP("UltimoAcesso") = Now.Date
            dRowP("Tabela") = ""
            dsParametros.Tables(0).Rows.Add(dRowP)
            Return dsParametros
        Else
            Dim Resultado As Boolean
            Resultado = ProcuraUtilizador(Utilizador, Codigo)
            If Resultado = False Then
                dsParametros = CriaDataSet()
                Return dsParametros 'retorna tabela parametros vazia
            Else
                Dim dRowUti As DataRow
                Dim IDstr As String
                Dim IDstrRSA As String
                Dim IDuti As Integer
                IDstr = GeraID() 'Gera ID aleatorio que identifica o utilizador durante a ligação
                dRowUti = dsUtilizadores.Tables(0).Rows(0)
                IDuti = dRowUti("ID")
                dsParametros = CriaDataSet()
                dRowP = dsParametros.Tables(0).NewRow()
                dRowP("IDutilizador") = IDstr
                dRowP("Nome") = dRowUti("Nome")
                dRowP("Empresa") = dRowUti("Empresa")
                dRowP("UltimoAcesso") = dRowUti("UltimoAcesso")
                dsParametros.Tables(0).Rows.Add(dRowP)
                dsTabelas = ProcuraTabelasUtilizador(IDuti)
            End If
        End If
    End Function
End Module
```

```

If dsTabelas.Tables(0).Rows.Count > 0 Then
    Dim dRowT As DataRow
    For Each dRowT In dsTabelas.Tables(0).Rows
        dRowP = dsParametros.Tables(0).NewRow()
        dRowP("IDbd") = dRowT("IDbd")
        dRowP("BaseDados") = dRowT("Descricao")
        dRowP("Tabela") = dRowT("Tabela")
        dRowP("Seleccionar") = dRowT("Seleccionar")
        dRowP("Inserir") = dRowT("Inserir")
        dRowP("Atualizar") = dRowT("Atualizar")
        dRowP("Apagar") = dRowT("Apagar")
        dsParametros.Tables(0).Rows.Add(dRowP)
    Next
End If
Dim Sucesso As Boolean
Dim strSQL As String
Dim dsLigHistorico As DataSet
strSQL = "SELECT *" & _
        " FROM LigacoesHistorico" & _
        " WHERE ID = 0"
dsLigHistorico = objDadoswsQL.CriaDataSet("LigacoesHistorico", strSQL)
Dim dRowLH As DataRow
dRowLH = dsLigHistorico.Tables(0).NewRow
dRowLH("IDutilizador") = IDuti
dRowLH("Inicio") = Now
dsLigHistorico.Tables(0).Rows.Add(dRowLH)
Sucesso = objDadoswsQL.UpdateDataSet(dsLigHistorico, "LigacoesHistorico")
Dim IDHistorico As Integer
strSQL = "SELECT *" & _
        " FROM LigacoesHistorico" & _
        " ORDER BY ID DESC"
dsLigHistorico = objDadoswsQL.CriaDataSet("LigacoesHistorico", strSQL)
IDHistorico = dsLigHistorico.Tables(0).Rows(0)("ID")
dRowUti("UltimoAcesso") = Now
Sucesso = objDadoswsQL.UpdateDataSet(dsUtilizadores, "Utilizadores")
Dim dsLigOnline As DataSet
strSQL = "SELECT *" & _
        " FROM LigacoesOnLine" & _
        " WHERE IDutilizador = " & IDuti
dsLigOnline = objDadoswsQL.CriaDataSet("LigacoesOnLine", strSQL)
Dim dRowLO As DataRow
If dsLigOnline.Tables(0).Rows.Count > 0 Then
    dRowLO = dsLigOnline.Tables(0).Rows(0)
    dRowLO("IDutilizadorTemp") = IDstr
    dRowLO("IDHistorico") = IDHistorico
Else
    dRowLO = dsLigOnline.Tables(0).NewRow
    dRowLO("IDutilizador") = IDuti
    dRowLO("IDutilizadorTemp") = IDstr
    dRowLO("IDHistorico") = IDHistorico
    dsLigOnline.Tables(0).Rows.Add(dRowLO)
End If
Sucesso = objDadoswsQL.UpdateDataSet(dsLigOnline, "LigacoesOnLine")
Return dsParametros
End If

End If
End Function

Function ProcuraUtilizador(ByVal Utilizador As String, _
                          ByVal Codigo As String) As Boolean
    Dim SQLstr As String
    SQLstr = "SELECT *" & _
            " FROM Utilizadores" & _
            " WHERE Utilizador = '" & Utilizador & "'" & _
            " AND Codigo = '" & Codigo & "'"
    dsUtilizadores = objDadoswsQL.CriaDataSet("Utilizadores", SQLstr)
    If dsUtilizadores.Tables(0).Rows.Count = 0 Then
        Return False
    Else
        Return True
    End If
End Function

```

```

Function CriaDataSet() As DataSet
    Dim Tabela As DataTable
    Dim dsVazia As New DataSet
    Tabela = New DataTable("Parametros")
    Tabela.Columns.Add("IDutilizador", Type.GetType("System.String"))
    Tabela.Columns.Add("Nome", Type.GetType("System.String"))
    Tabela.Columns.Add("Empresa", Type.GetType("System.String"))
    Tabela.Columns.Add("UltimoAcesso", Type.GetType("System.DateTime"))
    Tabela.Columns.Add("IDbd", Type.GetType("System.Int32"))
    Tabela.Columns.Add("BaseDados", Type.GetType("System.String"))
    Tabela.Columns.Add("Tabela", Type.GetType("System.String"))
    Tabela.Columns.Add("Seleccionar", Type.GetType("System.Boolean"))
    Tabela.Columns.Add("Inserir", Type.GetType("System.Boolean"))
    Tabela.Columns.Add("Atualizar", Type.GetType("System.Boolean"))
    Tabela.Columns.Add("Apagar", Type.GetType("System.Boolean"))
    dsVazia.Tables.Add(Tabela)
    Return dsVazia
End Function

Function GeraID() As String
    Dim I As Integer
    Dim strTEMP As String = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
    Dim IDstr As String
    Dim R As Integer
    Randomize()
    IDstr = ""
    For I = 1 To 20
        R = Int(Rnd() * 59) + 1
        IDstr &= strTEMP.Chars(R)
    Next
    Return IDstr
End Function

Function BuscaIDutilizador(ByVal IDtemp As String) As Integer
    Dim SQLstr As String
    Dim dsIDutilizador As DataSet
    SQLstr = "SELECT *" & _
        " FROM LigacoesOnline" & _
        " WHERE IDutilizadorTemp = '" & IDtemp & "'"
    dsIDutilizador = objDadoswsQL.CriaDataSet("IDutilizador", SQLstr)
    If dsIDutilizador.Tables(0).Rows.Count = 0 Then
        Return 0
    Else
        Return dsIDutilizador.Tables(0).Rows(0)("IDutilizador")
    End If
End Function

Function BuscaIDhistorico(ByVal IDtemp As String) As Integer
    Dim SQLstr As String
    Dim dsIDutilizador As DataSet
    SQLstr = "SELECT *" & _
        " FROM LigacoesOnline" & _
        " WHERE IDutilizadorTemp = '" & IDtemp & "'"
    dsIDutilizador = objDadoswsQL.CriaDataSet("IDutilizador", SQLstr)
    If dsIDutilizador.Tables(0).Rows.Count = 0 Then
        Return 0
    Else
        Return dsIDutilizador.Tables(0).Rows(0)("IDhistorico")
    End If
End Function

Function ProcuraTabelasUtilizador(ByVal IDuti As Integer) As DataSet
    Dim dsTab As DataSet
    Dim SQLstr As String
    SQLstr = "SELECT UtilizadoresPartilhas.IDbd, PartilhasBDs.Descricao," & _
        " UtilizadoresPartilhas.Tabela, UtilizadoresPartilhas.Seleccionar," & _
        " UtilizadoresPartilhas.Inserir, UtilizadoresPartilhas.Atualizar," & _
        " UtilizadoresPartilhas.Apagar" & _
        " FROM UtilizadoresPartilhas, PartilhasBDs" & _
        " WHERE UtilizadoresPartilhas.IDbd = PartilhasBDs.ID" & _
        " AND IDutilizador = " & IDuti & _
        " ORDER BY UtilizadoresPartilhas.IDbd"
    dsTab = objDadoswsQL.CriaDataSet("Tabelas", SQLstr)
    Return dsTab
End Function

```

```

Sub BuscaLigacao(ByVal IDtemp As String, _
    ByVal IDbd As Integer, _
    ByVal Tabela As String, _
    ByRef IDutilizador As Integer, _
    ByRef TipoLig As String, _
    ByRef ServidorSQL As String, _
    ByRef bdSQL As String, _
    ByRef AutenticacaoSQL As Integer, _
    ByRef UserIDSQ As String, _
    ByRef PasswordSQL As String, _
    ByRef CaminhoMDB As String, _
    ByRef BdMDB As String, _
    ByRef Conn As String, _
    ByRef Caminho As String)
    Dim SQLstr As String
    Dim dsLigacao As DataSet
    IDutilizador = BuscaIDutilizador(IDtemp)
    If IDutilizador = 0 Then
        Exit Sub
    End If
    SQLstr = "SELECT UtilizadoresPartilhas.*, PartilhasBDs.*" & _
        " FROM UtilizadoresPartilhas, PartilhasBDs" & _
        " WHERE UtilizadoresPartilhas.IDbd = PartilhasBDs.ID" & _
        " AND UtilizadoresPartilhas.IDbd = " & IDbd & _
        " AND UtilizadoresPartilhas.IDutilizador = " & IDutilizador & _
        " AND UtilizadoresPartilhas.Tabela = '" & Tabela & "'"
    dsLigacao = objDadoswsQL.CriaDataSet("Ligacao", SQLstr)
    If Not dsLigacao Is Nothing Then
        With dsLigacao.Tables(0)
            If .Rows.Count > 0 Then
                TipoLig = .Rows(0)("Tipo")
                Select Case TipoLig
                    Case "SQL"
                        ServidorSQL = .Rows(0)("ServidorSQL")
                        bdSQL = .Rows(0)("bdSQL")
                        AutenticacaoSQL = .Rows(0)("AutenticacaoSQL")
                        UserIDSQ = .Rows(0)("UserSQL")
                        PasswordSQL = .Rows(0)("ChaveSQL")
                    Case "MDB"
                        CaminhoMDB = .Rows(0)("CaminhoMDB")
                        BdMDB = .Rows(0)("BdMDB")
                End Select
            End If
        End With
    End If
    Select Case TipoLig
        Case "SQL"
            Conn = "Server=" & ServidorSQL & ";" & "Database=" & bdSQL
            Select Case AutenticacaoSQL
                Case 0 'integrada
                    Conn &= ";Integrated Security=SSPI"
                Case 1 'password
                    Conn &= ";user id= '" & UserIDSQ & "'"
                    Conn &= ";password = '" & PasswordSQL & "'"
            End Select
        Case "MDB"
            Caminho = CaminhoMDB & "\" & BdMDB & ".MDB"
        End Select
    End Sub

Sub BuscaLigacao(ByVal IDtemp As String, _
    ByVal IDbd As Integer, _
    ByRef IDutilizador As Integer, _
    ByRef TipoLig As String, _
    ByRef ServidorSQL As String, _
    ByRef bdSQL As String, _
    ByRef AutenticacaoSQL As Integer, _
    ByRef UserIDSQ As String, _
    ByRef PasswordSQL As String, _
    ByRef CaminhoMDB As String, _
    ByRef BdMDB As String, _
    ByRef Conn As String, _
    ByRef Caminho As String)
    Dim SQLstr As String
    Dim dsLigacao As DataSet
    IDutilizador = BuscaIDutilizador(IDtemp)

```

```

If IDutilizador = 0 Then
    Exit Sub
End If
SQLstr = "SELECT UtilizadoresPartilhas.*, PartilhasBDs.*" & _
        " FROM UtilizadoresPartilhas, PartilhasBDs" & _
        " WHERE UtilizadoresPartilhas.IDbd = PartilhasBDs.ID" & _
        " AND UtilizadoresPartilhas.IDbd = " & IDbd & _
        " AND UtilizadoresPartilhas.IDutilizador = " & IDutilizador
dsLigacao = objDadoswsQL.CriaDataSet("Ligacao", SQLstr)
If Not dsLigacao Is Nothing Then
    With dsLigacao.Tables(0)
        If .Rows.Count > 0 Then
            TipoLig = .Rows(0)("Tipo")
            Select Case TipoLig
                Case "SQL"
                    ServidorSQL = .Rows(0)("ServidorSQL")
                    bdSQL = .Rows(0)("bdSQL")
                    AutenticacaoSQL = .Rows(0)("AutenticacaoSQL")
                    UserIDSQL = .Rows(0)("UserSQL")
                    PasswordSQL = .Rows(0)("ChaveSQL")
                Case "MDB"
                    CaminhoMDB = .Rows(0)("CaminhoMDB")
                    BdMDB = .Rows(0)("BdMDB")
            End Select
        End If
    End With
End If

Select Case TipoLig
    Case "SQL"
        Conn = "Server=" & ServidorSQL & ";" & "Database=" & bdSQL
        Select Case AutenticacaoSQL
            Case 0 'integrada
                Conn &= ";Integrated Security=SSPI"
            Case 1 'password
                Conn &= ";user id= '" & UserIDSQL & "'"
                Conn &= ";password = '" & PasswordSQL & "'"
        End Select
    Case "MDB"
        Caminho = CaminhoMDB & "\" & BdMDB & ".MDB"
    End Select
End Sub

Sub GuardaOperacaoUtilizador(ByVal IDutilizador As Integer, _
                             ByVal TipoOperacao As String, _
                             ByVal BaseDados As String, _
                             ByVal Tabela As String, _
                             ByVal SucessoOp As Boolean)

    Dim strCampos As String
    Dim strValores As String
    Dim Data As Date
    If IDutilizador = 0 Then
        Exit Sub
    End If
    Data = DateTime.Now
    strCampos = "IDutilizador, Data, Operacao, Tabela, Sucesso"
    strValores = IDutilizador & _
                ", '" & Data & "'" & _
                ", '" & TipoOperacao & "'" & _
                ", '" & BaseDados & "'" & _
                ", '" & Tabela & "'" & _
                ", " & IIf(SucessoOp = True, 1, 0)

    Dim Sucesso As Boolean
    Sucesso = objDadoswsQL.ExecutaINSERT("HistoricoOperacoes", strCampos, strValores)
End Sub
End Module

```

C.3 Código do subsistema de Acesso a Dados

Exemplo C.3.1 – Código da camada de dados

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Data.OleDb
Imports System.Data.OracleClient
Imports System.Configuration
Imports System.Xml
Imports System.Xml.XPath
'''
' OLEDB
Public Class clsAcessoDadosOLEDB
    Public strConn As String
    Private msgErro As String
    Private msgErroOriginal As String

    Sub New()
        strConn = "Provider=Microsoft.Jet.OleDB.4.0;Data Source=" & ObtemCaminho()
    End Sub

    Sub New(ByVal Caminho As String)
        strConn = "Provider=Microsoft.Jet.OleDB.4.0;Data Source=" & Caminho
    End Sub

    Public Property MostraErro() As String
        Get
            MostraErro = msgErro
        End Get
        Set(ByVal Value As String)
            msgErro = Value
        End Set
    End Property

    Public Property MostraErroOriginal() As String
        Get
            MostraErroOriginal = msgErroOriginal
        End Get
        Set(ByVal Value As String)
            msgErroOriginal = Value
        End Set
    End Property

    Public Function ObtemCaminho() As String
        Const FicheiroXML As String = "C:\Inetpub\wwwroot\wsQL\bin\wsQL.xml"
        Dim xPathDoc As XPathDocument
        Dim Caminho As String
        Try
            xPathDoc = New XPathDocument(FicheiroXML)
            Dim xPathNav As XPathNavigator = xPathDoc.CreateNavigator()
            Dim oIter As XPathNodeIterator = xPathNav.Select("Configuracao/BaseDados/@Ligacao")
            oIter.MoveNext()
            Caminho = oIter.Current.Value
        Catch myErr As System.Exception
            msgErro = "Ficheiro SISLAweb.XML não encontrado!"
            msgErroOriginal = myErr.ToString
        End Try
        Return Caminho
    End Function

    Public Function AbreBaseDados() As OleDbConnection
        Dim Conn As OleDbConnection
        Conn = New OleDbConnection(strConn)
        Try
            Conn.Open()
        Catch myErr As System.Exception
            msgErro = "Servidor ou Base de Dados Inexistente!"
            msgErroOriginal = myErr.ToString
        End Try
        Return Conn
    End Function
End Class
```

```

Public Function CriaDataSet(ByVal Tabela As String) As DataSet
    Dim Conn As OleDbConnection
    Dim dAdapter As OleDbDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = "SELECT * FROM " & Tabela
    Conn = AbreBaseDados()
    Try
        dAdapter = New OleDbDataAdapter(sqlString, Conn)
        dSet = New DataSet
        dAdapter.Fill(dSet, Tabela)
        Conn.Close()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
        Return Nothing
    End Try
    Return dSet
End Function

Public Function CriaDataSet(ByVal Tabela As String, ByVal InstrucaoSQL As String) As DataSet
    Dim Conn As OleDbConnection
    Dim dAdapter As OleDbDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = InstrucaoSQL
    Conn = AbreBaseDados()
    Try
        dAdapter = New OleDbDataAdapter(sqlString, Conn)
        dSet = New DataSet
        dAdapter.Fill(dSet, Tabela)
        Conn.Close()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
        Return Nothing
    End Try
    Return dSet
End Function

Public Function CriaDataAdapter(ByVal Tabela As String) As OleDbDataAdapter
    Dim Conn As OleDbConnection
    Dim dAdapter As OleDbDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = "SELECT * FROM " & Tabela
    Conn = New OleDbConnection(strConn)
    dAdapter = New OleDbDataAdapter(sqlString, Conn)
    Return dAdapter
End Function

Public Function ExecutaINSERT(ByVal Tabela As String, ByVal Campos As String, ByVal Valores As
                                String) As Boolean
    Dim Conn As OleDbConnection
    Dim Cmd As New OleDbCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "INSERT INTO " & Tabela
    sqlString &= "(" & Campos & ")"
    sqlString &= " VALUES "
    sqlString &= "(" & Valores & ")"
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With
    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

```

```

Public Function ExecutaUPDATE(ByVal Tabela As String, _
                             ByVal Valores As String, _
                             ByVal SQLcampos As String) As Boolean

    Dim Conn As OleDbConnection
    Dim Cmd As New OleDbCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "UPDATE " & Tabela
    sqlString &= " SET " & Valores
    sqlString &= " WHERE " & SQLcampos
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With
    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

Public Function ExecutaDELETE(ByVal Tabela As String, ByVal SQLcampos As String) As Boolean
    Dim Conn As OleDbConnection
    Dim Cmd As New OleDbCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "DELETE FROM " & Tabela
    sqlString &= " WHERE " & SQLcampos
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With
    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

Public Function UpdateDataSet(ByVal dsDataSet As DataSet, ByVal Tabela As String) As Boolean
    Dim Sucesso As Boolean = True
    Dim daAdapter As OleDbDataAdapter
    daAdapter = CriaDataAdapter(Tabela)
    Try
        Dim cdComando As New OleDbCommandBuilder(daAdapter)
        daAdapter.Update(dsDataSet, Tabela)
    Catch Erro As Exception
        msgErro = Erro.Message
        msgErroOriginal = Erro.Message
        Sucesso = False
    End Try
    Return Sucesso
End Function
End Class

```

```

'''
' SQL Server
Public Class clsAcessoDadosSQL
    Public strConn As String
    Private msgErro As String
    Private msgErroOriginal As String
    Private strCaminho As String

    Sub New()
        strCaminho = ObtemCaminho()
    End Sub
    Sub New(ByVal Caminho As String)
        strCaminho = Caminho
    End Sub

    Public Property MostraErro() As String
        Get
            MostraErro = msgErro
        End Get
        Set(ByVal Value As String)
            msgErro = Value
        End Set
    End Property

    Public Property MostraErroOriginal() As String
        Get
            MostraErroOriginal = msgErroOriginal
        End Get
        Set(ByVal Value As String)
            msgErroOriginal = Value
        End Set
    End Property

    Public Property Caminho() As String
        Get
            Caminho = strCaminho
        End Get
        Set(ByVal Value As String)
            strCaminho = Value
        End Set
    End Property

    Public Function ObtemCaminho() As String
        Const FicheiroXML As String = "C:\Inetpub\wwwroot\wsQL\bin\wsQL.xml"
        Dim xPathDoc As XPathDocument
        Dim Caminho As String
        Try
            xPathDoc = New XPathDocument(FicheiroXML)
            Dim xPathNav As XPathNavigator = xPathDoc.CreateNavigator()
            Dim oIter As XPathNodeIterator = xPathNav.Select("Configuracao/Servidor/@Ligacao")
            oIter.MoveNext()
            Caminho = oIter.Current.Value
        Catch myErr As System.Exception
            msgErro = "Ficheiro wsQL.XML não encontrado!"
            msgErroOriginal = myErr.ToString
        End Try
        Return Caminho
    End Function

    Public Function Ligacao() As SqlConnection
        Dim Conn As SqlConnection
        strConn = strCaminho
        Conn = New SqlConnection(strConn)
        Return Conn
    End Function

    Public Function AbreBaseDados() As SqlConnection
        Dim Conn As SqlConnection
        Conn = Ligacao()
        Try
            Conn.Open()
        Catch myErr As System.Exception
            msgErro = "Servidor ou Base de Dados Inexistente!"
            msgErroOriginal = myErr.ToString
        End Try
        Return Conn
    End Function

```

```

Public Function CriaDataAdapter(ByVal Tabela As String) As SqlDataAdapter
    Dim Conn As SqlConnection
    Dim dAdapter As SqlDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = "SELECT * FROM " & Tabela
    Conn = Ligacao()
    dAdapter = New SqlDataAdapter(sqlString, Conn)
    Return dAdapter
End Function

Public Function CriaDataSet(ByVal Tabela As String) As DataSet
    Dim Conn As SqlConnection
    Dim dAdapter As SqlDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = "SELECT * FROM " & Tabela
    Conn = AbreBaseDados()
    Try
        dAdapter = New SqlDataAdapter(sqlString, Conn)
        dSet = New DataSet
        dAdapter.Fill(dSet, Tabela)
        Conn.Close()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
        Return Nothing
    End Try
    Return dSet
End Function

Public Function CriaDataSet(ByVal Tabela As String, ByVal InstrucaoSQL As String) As DataSet
    Dim Conn As SqlConnection
    Dim dAdapter As SqlDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = InstrucaoSQL
    Conn = AbreBaseDados()
    Try
        dAdapter = New SqlDataAdapter(sqlString, Conn)
        dSet = New DataSet
        dAdapter.Fill(dSet, Tabela)
        Conn.Close()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
        Return Nothing
    End Try
    Return dSet
End Function

Public Function CriaDataView(ByVal Tabela As String, ByVal CampoSort As String) As DataView
    Dim dSet As DataSet
    Dim dView As DataView
    dSet = CriaDataSet(Tabela)
    dView = New DataView(dSet.Tables(0), "", CampoSort, DataViewRowState.OriginalRows)
    Return dView
End Function

Public Function CriaDataReader(ByVal InstrucaoSQL As String) As SqlDataReader
    Dim Conn As SqlConnection
    Dim dReader As SqlDataReader
    Dim sqlCommand As SqlCommand
    sqlCommand = New SqlCommand(InstrucaoSQL, Conn)
    Try
        Conn = AbreBaseDados()
        dReader = sqlCommand.ExecuteReader
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Return dReader
End Function

```

```

Public Function ExecutaINSERT(ByVal Tabela As String, ByVal Campos As String, ByVal
                               Valores As String) As Boolean

    Dim Conn As SqlConnection
    Dim Cmd As New SqlCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "INSERT INTO " & Tabela
    sqlString &= "(" & Campos & ")"
    sqlString &= " VALUES "
    sqlString &= "(" & Valores & ")"
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With
    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

Public Function ExecutaUPDATE(ByVal Tabela As String, _
                              ByVal Valores As String, _
                              ByVal SQLcampos As String) As Boolean

    Dim Conn As SqlConnection
    Dim Cmd As New SqlCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "UPDATE " & Tabela
    sqlString &= " SET " & Valores
    sqlString &= " WHERE " & SQLcampos
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With
    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

Public Function ExecutaDELETE(ByVal Tabela As String, ByVal SQLcampos As String) As Boolean

    Dim Conn As SqlConnection
    Dim Cmd As New SqlCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "DELETE FROM " & Tabela
    sqlString &= " WHERE " & SQLcampos
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With

```

```

    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

Public Function UpdateDataSet (ByVal dsDataSet As DataSet, ByVal Tabela As String) As Boolean
    Dim Sucesso As Boolean = True
    Dim daAdapter As SqlDataAdapter
    daAdapter = CriaDataAdapter(Tabela)
    Try
        Dim cdComando As New SqlCommandBuilder(daAdapter)
        daAdapter.Update(dsDataSet, Tabela)
    Catch Erro As Exception
        msgErro = Erro.Message
        msgErroOriginal = Erro.Message
        Sucesso = False
    End Try
    Return Sucesso
End Function
End Class

'''
' ORACLE
Public Class clsAcessoDadosORACLE
    Public strConn As String
    Private msgErro As String
    Private msgErroOriginal As String
    Private strCaminho As String

    Sub New()
        strCaminho = ObtemCaminho()
    End Sub

    Sub New(ByVal Caminho As String)
        strCaminho = Caminho
    End Sub

    Public Property MostraErro() As String
        Get
            MostraErro = msgErro
        End Get
        Set(ByVal Value As String)
            msgErro = Value
        End Set
    End Property

    Public Property MostraErroOriginal() As String
        Get
            MostraErroOriginal = msgErroOriginal
        End Get
        Set(ByVal Value As String)
            msgErroOriginal = Value
        End Set
    End Property

    Public Property Caminho() As String
        Get
            Caminho = strCaminho
        End Get
        Set(ByVal Value As String)
            strCaminho = Value
        End Set
    End Property
End Class

```

```

Public Function ObtemCaminho() As String
    Const FicheiroXML As String = "C:\Inetpub\wwwroot\wsQL\bin\wsQL.xml"
    Dim xPathDoc As XPathDocument
    Dim Caminho As String
    Try
        xPathDoc = New XPathDocument(FicheiroXML)
        Dim xPathNav As XPathNavigator = xPathDoc.CreateNavigator()
        Dim oIter As XPathNodeIterator = xPathNav.Select("Configuracao/Servidor/@Ligacao")
        oIter.MoveNext()
        Caminho = oIter.Current.Value
    Catch myErr As System.Exception
        msgErro = "Ficheiro wsQL.XML não encontrado!"
        msgErroOriginal = myErr.ToString
    End Try
    Return Caminho
End Function

Public Function Ligacao() As OracleConnection
    Dim Conn As OracleConnection
    strConn = strCaminho
    Conn = New OracleConnection(strConn)
    Return Conn
End Function

Public Function AbreBaseDados() As OracleConnection
    Dim Conn As OracleConnection
    Conn = Ligacao()
    Try
        Conn.Open()
    Catch myErr As System.Exception
        msgErro = "Servidor ou Base de Dados Inexistente!"
        msgErroOriginal = myErr.ToString
    End Try
    Return Conn
End Function

Public Function CriaDataAdapter(ByVal Tabela As String) As OracleDataAdapter
    Dim Conn As OracleConnection
    Dim dAdapter As OracleDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = "SELECT * FROM " & Tabela
    Conn = Ligacao()
    dAdapter = New OracleDataAdapter(sqlString, Conn)
    Return dAdapter
End Function

Public Function CriaDataSet(ByVal Tabela As String) As DataSet
    Dim Conn As OracleConnection
    Dim dAdapter As OracleDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = "SELECT * FROM " & Tabela
    Conn = AbreBaseDados()
    Try
        dAdapter = New OracleDataAdapter(sqlString, Conn)
        dSet = New DataSet
        dAdapter.Fill(dSet, Tabela)
        Conn.Close()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
        Return Nothing
    End Try
    Return dSet
End Function

```

```

Public Function CriaDataSet(ByVal Tabela As String, ByVal InstrucaoSQL As String) As DataSet
    Dim Conn As OracleConnection
    Dim dAdapter As OracleDataAdapter
    Dim dSet As DataSet
    Dim sqlString As String = InstrucaoSQL
    Conn = AbreBaseDados()
    Try
        dAdapter = New OracleDataAdapter (sqlString, Conn)
        dSet = New DataSet
        dAdapter.Fill(dSet, Tabela)
        Conn.Close()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
        Return Nothing
    End Try
    Return dSet
End Function

Public Function CriaDataView(ByVal Tabela As String, ByVal CampoSort As String) As DataView
    Dim dSet As DataSet
    Dim dView As DataView
    dSet = CriaDataSet(Tabela)
    dView = New DataView(dSet.Tables(0), "", CampoSort, DataViewRowState.OriginalRows)
    Return dView
End Function

Public Function CriaDataReader(ByVal InstrucaoSQL As String) As OracleDataReader
    Dim Conn As OracleConnection
    Dim dReader As OracleDataReader
    Dim sqlCmd As OracleCommand
    sqlCmd = New OracleCommand(InstrucaoSQL, Conn)
    Try
        Conn = AbreBaseDados()

        dReader = sqlCmd.ExecuteReader
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Return dReader
End Function

Public Function ExecutaINSERT(ByVal Tabela As String, ByVal Campos As String, ByVal
                               Valores As String) As Boolean
    Dim Conn As OracleConnection
    Dim Cmd As New OracleCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "INSERT INTO " & Tabela
    sqlString &= "(" & Campos & ")"
    sqlString &= " VALUES "
    sqlString &= "(" & Valores & ")"
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With
    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

```

```

Public Function ExecutaUPDATE(ByVal Tabela As String, _
                             ByVal Valores As String, _
                             ByVal SQLcampos As String) As Boolean
    Dim Conn As OracleConnection
    Dim Cmd As New OracleCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "UPDATE " & Tabela
    sqlString &= " SET " & Valores
    sqlString &= " WHERE " & SQLcampos
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With
    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

Public Function ExecutaDELETE(ByVal Tabela As String, ByVal SQLcampos As String) As Boolean
    Dim Conn As OracleConnection
    Dim Cmd As New OracleCommand
    Dim sqlString As String = ""
    Dim nRegs As Integer
    sqlString = "DELETE FROM " & Tabela
    sqlString &= " WHERE " & SQLcampos
    Conn = AbreBaseDados()
    With Cmd
        .Connection = Conn
        .CommandTimeout = 0
        .CommandText = sqlString
        .CommandType = CommandType.Text
    End With
    Try
        nRegs = Cmd.ExecuteNonQuery()
    Catch myErr As System.Exception
        msgErroOriginal = myErr.ToString
    End Try
    Conn.Close()
    If nRegs > 0 Then
        Return True
    Else
        Return False
    End If
End Function

Public Function UpdateDataSet(ByVal dsDataSet As DataSet, ByVal Tabela As String) As Boolean
    Dim Sucesso As Boolean = True
    Dim daAdapter As OracleDataAdapter
    daAdapter = CriaDataAdapter(Tabela)
    Try
        Dim cdComando As New OracleCommandBuilder(daAdapter)
        daAdapter.Update(dsDataSet, Tabela)
    Catch Err As Exception
        msgErro = Err.Message
        msgErroOriginal = Err.Message
        Sucesso = False
    End Try
    Return Sucesso
End Function
End Class

```

Documento WSDL do Web Service wsQL

Exemplo D.1 – Documento WSDL do Web Service wsQL

```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://fe.up.pt/wsQL/wsQL"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://fe.up.pt/wsQL/wsQL"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://fe.up.pt/wsQL/wsQL">
      <s:import namespace="http://www.w3.org/2001/XMLSchema" />
      <s:element name="GetChavePublica">
        <s:complexType />
      </s:element>
      <s:element name="GetChavePublicaResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetChavePublicaResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="Login">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Utilizador" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Codigo" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="LoginResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="LoginResult">
              <s:complexType>
                <s:sequence>
                  <s:element ref="s:schema" />
                  <s:any />
                </s:sequence>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="LoginComChavePublicaCliente">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Utilizador" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Codigo" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="ChavePublicaCliente" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>

```

```

</s:complexType>
</s:element>
<s:element name="LoginComChavePublicaClienteResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="LoginComChavePublicaClienteResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="FechaSessao">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="IDtemp" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="FechaSessaoResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="FechaSessaoResult" type="s:boolean"/>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetEstruturaTabela">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="IDtemp" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="IDbd" type="s:int" />
      <s:element minOccurs="0" maxOccurs="1" name="Tabela" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetEstruturaTabelaResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetEstruturaTabelaResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetDadosTabela">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="IDtemp" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="IDbd" type="s:int" />
      <s:element minOccurs="0" maxOccurs="1" name="Tabela" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetDadosTabelaResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetDadosTabelaResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="ExecutaSQL">

```

```

<s:complexType>
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="IDtemp" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="IDbd" type="s:int" />
    <s:element minOccurs="0" maxOccurs="1" name="strSQL" type="s:string" />
  </s:sequence>
</s:complexType>
</s:element>
<s:element name="ExecutaSQLResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="ExecutaSQLResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="EscreveDadosTabela">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="IDtemp" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="Tabela" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="EscreveDadosTabelaResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="EscreveDadosTabelaResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="HistoricoLigacoes">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="IDtemp" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="HistoricoLigacoesResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="HistoricoLigacoesResult">
        <s:complexType>
          <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="HistoricoOperacoes">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="IDtemp" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="HistoricoOperacoesResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="HistoricoOperacoesResult">
        <s:complexType>

```

```

        <s:sequence>
            <s:element ref="s:schema" />
            <s:any />
        </s:sequence>
    </s:complexType>
</s:element>
</s:sequence>
</s:complexType>
</s:element>
</s:schema>
</wsdl:types>
<wsdl:message name="GetChavePublicaSoapIn">
    <wsdl:part name="parameters" element="tns:GetChavePublica" />
</wsdl:message>
<wsdl:message name="GetChavePublicaSoapOut">
    <wsdl:part name="parameters" element="tns:GetChavePublicaResponse" />
</wsdl:message>
<wsdl:message name="LoginSoapIn">
    <wsdl:part name="parameters" element="tns:Login" />
</wsdl:message>
<wsdl:message name="LoginSoapOut">
    <wsdl:part name="parameters" element="tns:LoginResponse" />
</wsdl:message>
<wsdl:message name="LoginComChavePublicaClienteSoapIn">
    <wsdl:part name="parameters" element="tns:LoginComChavePublicaCliente" />
</wsdl:message>
<wsdl:message name="LoginComChavePublicaClienteSoapOut">
    <wsdl:part name="parameters" element="tns:LoginComChavePublicaClienteResponse" />
</wsdl:message>
<wsdl:message name="FechaSessaoSoapIn">
    <wsdl:part name="parameters" element="tns:FechaSessao" />
</wsdl:message>
<wsdl:message name="FechaSessaoSoapOut">
    <wsdl:part name="parameters" element="tns:FechaSessaoResponse" />
</wsdl:message>
<wsdl:message name="GetEstruturaTabelaSoapIn">
    <wsdl:part name="parameters" element="tns:GetEstruturaTabela" />
</wsdl:message>
<wsdl:message name="GetEstruturaTabelaSoapOut">
    <wsdl:part name="parameters" element="tns:GetEstruturaTabelaResponse" />
</wsdl:message>
<wsdl:message name="GetDadosTabelaSoapIn">
    <wsdl:part name="parameters" element="tns:GetDadosTabela" />
</wsdl:message>
<wsdl:message name="GetDadosTabelaSoapOut">
    <wsdl:part name="parameters" element="tns:GetDadosTabelaResponse" />
</wsdl:message>
<wsdl:message name="ExecutaSQLSoapIn">
    <wsdl:part name="parameters" element="tns:ExecutaSQL" />
</wsdl:message>
<wsdl:message name="ExecutaSQLSoapOut">
    <wsdl:part name="parameters" element="tns:ExecutaSQLResponse" />
</wsdl:message>
<wsdl:message name="EscreveDadosTabelaSoapIn">
    <wsdl:part name="parameters" element="tns:EscreveDadosTabela" />
</wsdl:message>
<wsdl:message name="EscreveDadosTabelaSoapOut">
    <wsdl:part name="parameters" element="tns:EscreveDadosTabelaResponse" />
</wsdl:message>
<wsdl:message name="HistoricoLigacoesSoapIn">
    <wsdl:part name="parameters" element="tns:HistoricoLigacoes" />
</wsdl:message>
<wsdl:message name="HistoricoLigacoesSoapOut">
    <wsdl:part name="parameters" element="tns:HistoricoLigacoesResponse" />
</wsdl:message>
<wsdl:message name="HistoricoOperacoesSoapIn">
    <wsdl:part name="parameters" element="tns:HistoricoOperacoes" />
</wsdl:message>
<wsdl:message name="HistoricoOperacoesSoapOut">
    <wsdl:part name="parameters" element="tns:HistoricoOperacoesResponse" />
</wsdl:message>
<wsdl:portType name="wsQLSoap">
    <wsdl:operation name="GetChavePublica">
        <wsdl:input message="tns:GetChavePublicaSoapIn" />
        <wsdl:output message="tns:GetChavePublicaSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="Login">

```

```

    <wsdl:input message="tns:LoginSoapIn" />
    <wsdl:output message="tns:LoginSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="LoginComChavePublicaCliente">
    <wsdl:input message="tns:LoginComChavePublicaClienteSoapIn" />
    <wsdl:output message="tns:LoginComChavePublicaClienteSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="FechaSessao">
    <wsdl:input message="tns:FechaSessaoSoapIn" />
    <wsdl:output message="tns:FechaSessaoSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetEstruturaTabela">
    <wsdl:input message="tns:GetEstruturaTabelaSoapIn" />
    <wsdl:output message="tns:GetEstruturaTabelaSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetDadosTabela">
    <wsdl:input message="tns:GetDadosTabelaSoapIn" />
    <wsdl:output message="tns:GetDadosTabelaSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="ExecutaSQL">
    <wsdl:input message="tns:ExecutaSQLSoapIn" />
    <wsdl:output message="tns:ExecutaSQLSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="EscreveDadosTabela">
    <wsdl:input message="tns:EscreveDadosTabelaSoapIn" />
    <wsdl:output message="tns:EscreveDadosTabelaSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="HistoricoLigacoes">
    <wsdl:input message="tns:HistoricoLigacoesSoapIn" />
    <wsdl:output message="tns:HistoricoLigacoesSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="HistoricoOperacoes">
    <wsdl:input message="tns:HistoricoOperacoesSoapIn" />
    <wsdl:output message="tns:HistoricoOperacoesSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="wsQLSoap" type="tns:wsQLSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="GetChavePublica">
    <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/GetChavePublica" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Login">
    <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/Login" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="LoginComChavePublicaCliente">
    <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/LoginComChavePublicaCliente" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="FechaSessao">
    <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/FechaSessao" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetEstruturaTabela">
    <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/GetEstruturaTabela" style="document" />
    <wsdl:input>

```

```

    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetDadosTabela">
  <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/GetDadosTabela" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="ExecutaSQL">
  <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/ExecutaSQL" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="EscreveDadosTabela">
  <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/EscreveDadosTabela" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="HistoricoLigacoes">
  <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/HistoricoLigacoes" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="HistoricoOperacoes">
  <soap:operation soapAction="http://fe.up.pt/wsQL/wsQL/HistoricoOperacoes" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="wsQL">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
  <wsdl:port name="wsQLSoap" binding="tns:wsQLSoap">
    <soap:address location="http://localhost/wsQL/wsQL.asmx" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```
