**UNIVERSIDADE DO VALE DO ITAJAÍ**
**CENTRO DE EDUCAÇÃO SUPERIOR DE**
**CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR**
**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

# PROPOSTA DE UM PADRÃO PARA A COMUNICAÇÃO DE DADOS EM SISTEMAS DE TELEMETRIA BASEADO EM WEB SERVICES

Área de Redes de Computadores

José Morelli Neto

Itajaí (SC), Dezembro de 2003.

**UNIVERSIDADE DO VALE DO ITAJAÍ**
**CENTRO DE EDUCAÇÃO SUPERIOR DE**
**CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR**
**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

# PROPOSTA DE UM PADRÃO PARA A COMUNICAÇÃO DE DADOS EM SISTEMAS DE TELEMETRIA BASEADO EM WEB SERVICES

Área de Redes de Computadores

José Morelli Neto

Relatório apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Ciência da Computação para análise e aprovação.

Itajaí (SC), Dezembro de 2003.

# EQUIPE TÉCNICA

**Acadêmico**

José Morelli Neto

**Professor Orientador**

Rodrigo Becke Cabral, Dr.

**Coordenadores dos Trabalhos de Conclusão de Curso**

Anita Maria da Rocha Fernandes, Dra

Cesar Albenes Zeferino, Dr.

**Coordenador do Curso**

Luís Carlos Martins, Esp.

# DEDICATÓRIA

*Dedico este trabalho aos meus pais, Jesenir e Renato, que sempre acreditaram na minha capacidade e são meus maiores incentivadores.*

# AGRADECIMENTOS

Agradeço a **Deus** por ter proporcionado a minha existência, a saúde e maturidade para hoje poder estar finalizando esse trabalho.

À minha mãe, pelo apoio e incentivo durante toda essa caminhada, principalmente nos meus momentos de impaciência.

Aos meus irmãos, Fabiano e Fabíola, pelo apoio dado durante toda minha vida.

À minha namorada Andréa, que foi a minha maior incentivadora para concluir esse curso, dando-me força e carinho nos momentos difíceis, apoiando-me e compreendendo a minha ausência devido ao trabalho e aos estudos.

Aos familiares que me ajudaram em momentos difíceis, em especial ao meu cunhado Silvio.

A todos os amigos e colegas que me acompanharam na fase mais importante da minha vida, ensinando-me ou compartilhando momentos difíceis, (Anelise, Fabricio Bortoluzzi, Fernando Gomes, Jesiel da Silva, Luciene Ferrão, Marcel Kohls, Marcio Ota, Mateus Pelloso, Rafael Canan, e muitos outros... a lista é extensa :) ).

Aos companheiros de trabalho, que por muitas vezes “seguraram as pontas”, me permitindo finalizar trabalhos importantes (Emílio Erthal, Jorge Limas, Leonardo Peixoto).

Aos mestres, pelo conhecimento repassado, apoio e por acreditarem na minha capacidade (Ademir Goulart, Anita, Gilberto Luy, Ovídio, Rafael Cancian Rodrigo Cabral e outros).

Resumindo: muito obrigado a todos... e a lista é por ordem alfabética, e não por importância!

Neto.

# SUMÁRIO

**LISTA DE ABREVIATURAS E SIGLAS ..... ix**

**LISTA DE FIGURAS ..... xi**

**LISTA DE TABELAS ..... xiii**

**RESUMO ..... xiv**

**ABSTRACT ..... xv**

**I - INTRODUÇÃO ..... 1**

**1 APRESENTAÇÃO ..... 1**

**2 JUSTIFICATIVA ..... 2**

**3 IMPORTÂNCIA ..... 3**

**4 OBJETIVOS DO TRABALHO ..... 4**

**4.1 Objetivo Geral ..... 4**

**4.2 Objetivos Específicos ..... 4**

**5 METODOLOGIA ..... 4**

**II - REVISÃO BIBLIOGRÁFICA ..... 7**

**1 TELEMETRIA ..... 7**

**1.1 Introdução ..... 7**

**1.2 Considerações ..... 10**

**2 PADRÃO ..... 11**

**2.1 Introdução ..... 11**

**2.2 Definição de Padrão ..... 11**

**2.3 A importância dos Padrões ..... 12**

**2.4 Normalização ..... 12**

**2.5 Entidades de Normalização ..... 13**
2.5.1 ISO (International Organization for Standardization) ..... 13
2.5.2 ABNT (Associação Brasileira de Normas Técnicas) ..... 13
2.5.3 CMN (Comitê Mercosul de Normalização) ..... 13
2.5.4 IEC (International Electrotechnical Comission) ..... 14
2.5.5 NMEA (National Marine Electronics Association) ..... 14
2.5.6 W3C (World Wide Web Consortium) ..... 14
**2.6 Exemplos de Padrões ..... 15**
**2.7 Construção de Padrões ..... 15**
**2.8 Considerações ..... 17**
**3 WEB SERVICES ..... 17**
**3.1 Introdução ..... 17**
**3.2 Vantagens da Utilização de Web Services ..... 19**
**3.3 XML ..... 20**
3.3.1 Namespaces ..... 20
3.3.2 XML Schema ..... 21
3.3.2.1 Tipos de Dados Simples e Complexos ..... 21
**3.4 UDDI ..... 22**
3.4.1 Função da UDDI ..... 23
**3.5 WSDL ..... 23**
3.5.1 Introdução ..... 23
3.5.2 Estrutura de um documento WSDL ..... 25
3.5.2.1 Definitions ..... 26
3.5.2.2 Types ..... 27
3.5.2.3 Message ..... 28
3.5.2.4 portType e Operation ..... 29
3.5.2.5 Binding ..... 31
3.5.2.5.1 soap:binding ..... 32

3.5.2.5.2 soap:operation ..... 33
3.5.2.5.3 soap:body ..... 33
3.5.2.6 Service e Port ..... 33
**3.6 Considerações ..... 34**
**4 PROTOCOLOS DE WEB SERVICES ..... 34**
**4.1 Introdução ..... 34**
**4.2 XML-RPC ..... 34**
4.2.1 Arquitetura do XML-RPC ..... 35
4.2.2 Tipos de Dados ..... 35
4.2.3 Formato da Requisição ..... 36
4.2.4 Formato da Resposta ..... 37
**4.3 SOAP ..... 38**
4.3.1 Arquitetura ..... 39
4.3.2 Mensagem SOAP ..... 40
4.3.2.1 Envelope ..... 41
4.3.2.2 Cabeçalho (Header) ..... 41
4.3.2.2.1 Atributo Actor ..... 42
4.3.2.2.2 Atributo mustUnderstand ..... 42
4.3.2.3 Corpo (Body) ..... 43
4.3.2.4 Falha (Fault) ..... 44
4.3.3 Tipos de Dados ..... 44
4.3.3.1 Tipos Simples ..... 45
4.3.3.1.1 Strings ..... 45
4.3.3.1.2 Enumerations ..... 45
4.3.3.1.3 Array de Bytes ..... 46
4.3.3.2 Tipos Complexos ..... 46
4.3.3.2.1 Estruturas ..... 46
4.3.3.2.2 Arrays ..... 47
4.3.4 Encapsulando mensagens SOAP em HTTP ..... 48

4.3.4.1 Requisição HTTP SOAP ..... 48
4.3.4.2 Resposta HTTP SOAP ..... 49
**4.4 Comparativo entre XML-RPC e SOAP ..... 50**
**4.5 Segurança ..... 51**
4.5.1 Autenticação ..... 51
4.5.2 Autorização ..... 52
4.5.3 Não Repudiação ..... 52
4.5.4 Integridade ..... 52
4.5.5 Privacidade ..... 52
**4.6 Considerações ..... 53**
**5 RASTRO ..... 53**
**5.1 Introdução ..... 53**
**5.2 Arquitetura do Sistema ..... 55**
5.2.1 Recepção de Dados ..... 55
5.2.2 Mapa de navegação via Web ..... 56
**5.3 Considerações ..... 58**
**III - DESENVOLVIMENTO ..... 60**
**1 INTRODUÇÃO ..... 60**
**2 MODELAGEM DO SISTEMA ..... 60**
**2.1 Diagrama de Caso de Uso ..... 60**
**2.2 Modelo de Informação ..... 64**
**3 DESCRIÇÃO DO PADRÃO PROPOSTO ..... 66**
**4 IMPLEMENTAÇÃO DO PADRÃO PROPOSTO ..... 67**
**4.1 Introdução ..... 67**
**4.2 Tecnologia ..... 67**
4.2.1 PHP ..... 67
4.2.2 PEAR ..... 68

4.2.3 MySQL ..... 68
4.2.4 Oracle ..... 68
**4.3 Validação do Padrão Proposto ..... 69**
4.3.1 Protótipo ..... 69
4.3.1.1 Módulo Servidor ..... 70
4.3.1.2 Módulo Cliente ..... 70
4.3.2 Implementação do Padrão no Sistema RASTRO ..... 73
**IV - CONCLUSÕES E RECOMENDAÇÕES ..... 76**
**BIBLIOGRAFIA ..... 79**
**ANEXOS ..... 83**

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ABNT | Associação Brasileira de Normas Técnicas |
| ANATEL | Agência Nacional de Telecomunicações |
| CMN | Comitê Mercosul de Normalização |
| CSMA | *Carrier Sense Multiple Access* |
| DCOM | *Distributed Component Object Model* |
| DIS | *Draft International Standard* |
| FTP | *File Transfer Protocol* |
| GPS | *Global Position System* |
| HTML | *HyperText Markup Language* |
| HTTP | *HyperText Transfer Protocol* |
| HTTPS | *HyperText Transfer Protocol Secure* |
| IEC | *International Electrotechnical Comission* |
| IETF | *Internet Engineering Task Force* |
| IIOP | *Internet Inter-Operability Protocol* |
| IP | *Internet Protocol* |
| ISO | *International Organization for Standardization* |
| MHz | *Mega Hertz* |
| MIME | *Multi-purpose Internet Mail Extension* |
| NMEA | *National Marine Electronics Association* |
| OGC | *Open Gis Consortium* |
| PEAR | *PHP Extension and Application Repository* |
| PHP | *PHP: Hypertext Preprocessor* |
| PKI | *Public Key Infrastructure* |
| RMI | *Remote Method Invocation* |
| RPC | *Remote Procedure Call* |
| SAML | *Security Assertion Markup Language* |
| SGBD | Sistema de Gerenciamento de Bando de Dados |
| SGML | *Standard Generalized Markup Language* |
| SMTP | *Simple Mail Transport Protocol* |
| SOAP | *Simple Object Access Protocol* |
| SQL | *Structured Query Language* |
| SSL | *Secure Socket Layer* |
| UDDI | *Universal Description, Discovery and Integration* |
| UHF | *Ultra High Frequency* |

| | |
|---|---|
| UML | *Unified Modeling Language* |
| URI | *Uniform Resource Identifiers* |
| URL | *Uniform Resource Locator* |
| VHF | *Very High Frequency* |
| XAML | *Extensible Access Control List* |
| XKMS | *XML Key Management Specification* |
| XML | *Extensible Markup Language* |
| XSD | *XML Schema Document* |
| W3C | *World Wide Web Consortium* |
| WSDL | *Web Services Description Language* |

# LISTA DE FIGURAS

Figura 1: Funcionamento Simplificado de um *Web Service*. ..... 18
Figura 2: Funcionamento do WSDL. ..... 25
Figura 3: Estrutura de um arquivo WSDL. ..... 26
Figura 4: Exemplo do elemento `definitions` de um arquivo WSDL. ..... 27
Figura 5: Exemplo de uso do elemento `type`. ..... 28
Figura 6: Exemplos do elemento `message`. ..... 28
Figura 7: Exemplo de um elemento `message` com atributo `type`. ..... 29
Figura 8: Exemplo de uso dos elementos `portType` e `operation`. ..... 30
Figura 9: Formas de operação suportadas pelo WSDL. ..... 31
Figura 10: Exemplo de uso do elemento `binding`. ..... 32
Figura 11: Exemplo de uso dos elementos `service` e `port`. ..... 33
Figura 12: Exemplo de uma solicitação XML-RPC. ..... 36
Figura 13: Exemplo de uma resposta XML-RPC. ..... 37
Figura 14: Exemplo de uma resposta XML-RPC com falha. ..... 38
Figura 15: Composição de uma mensagem SOAP. ..... 40
Figura 16: Exemplo do elemento cabeçalho com uso de atributos. ..... 41
Figura 17: Exemplo de uso dos atributos `actor` e `mustUnderstand`. ..... 43
Figura 18: Exemplo do elemento corpo. ..... 43
Figura 19: Exemplo de representação de tipos simples em um único elemento. ..... 45
Figura 20: Exemplo de representação de tipos simples em *schema*. ..... 45
Figura 21: Exemplo de uso do tipo simples `enumeration`. ..... 46
Figura 22: Exemplo de um XML *Schema* para a estrutura estudante. ..... 47
Figura 23: Exemplo de uso da estrutura estudante. ..... 47
Figura 24: Exemplo de uso do tipo *Array*. ..... 48
Figura 25: Exemplo de um HTTP SOAP utilizando POST. ..... 49
Figura 26: Exemplo de resposta HTTP com erro. ..... 49
Figura 27: Arquitetura de funcionamento do sistema RASTRO. ..... 54
Figura 28: Navegação em mapas via Web – últimas posições. ..... 56
Figura 29: Pontos rastreados de uma embarcação. ..... 57

Figura 30: Diagrama de Caso do padrão proposto. 61
Figura 31: Caso de Uso Registrar Ocorrência. 62
Figura 32: Caso de Uso Solicitar Pontos de Coleta. 62
Figura 33: Caso de Uso Solicitar Sensores. 63
Figura 34: Caso de Uso Solicitar Ocorrências. 63
Figura 35: Modelo ER lógico dos dados. 64
Figura 36: Modelo ER físico dos dados. 64
Figura 37: Tela inicial do Módulo Cliente. 71
Figura 38: Tela de inserção de ocorrências. 71
Figura 39: Tela de consultas do Módulo Cliente. 72
Figura 40: Funcionamento inicial do módulo de Recepção de Dados. 73
Figura 41: Alteração efetuada no módulo de Recepção de Dados. 74

# LISTA DE TABELAS

Tabela 1: Tipos de dados suportados pelo XML-RPC. 35
Tabela 2: Comparativo entre algumas características do XML-RPC e do SOAP. 51
Tabela 3: Descrição das tabelas de dados utilizadas no sistema 65
Tabela 4: Dicionário de Dados da tabela Cliente 65
Tabela 5: Dicionário de Dados da tabela Coleta 65
Tabela 6: Dicionário de Dados da tabela Sensor 66
Tabela 7: Dicionário de Dados da tabela SenCod 66
Tabela 8: Dicionário de Dados da tabela Registro 66
Tabela 9: Dicionário de Dados da tabela Leitura 66

# RESUMO

Telemetria é a técnica que mede quantidades, transmitindo os resultados para um centro de controle que é responsável pela interpretação, apresentação e/ou armazenamento dos valores medidos. Os meios de transmissão de dados utilizados podem ser muito variados, abrangendo desde linhas telefônicas convencionais até a transmissão por meio da Internet. A forma de transação dos dados de telemetria tem sido padronizada por diversas organizações com o objetivo de tornar as indústrias que fabricam equipamentos de telemetria compatíveis entre si. Com o advento de novas tecnologias, existe uma lacuna na padronização das transações de telemetria em nível de camada de aplicação. Este trabalho visa preencher esta lacuna com a proposta de um protocolo de transações de dados de telemetria através da tecnologia de *Web Services*. Essa normalização tem aplicação imediata no sistema de rastreamento de embarcações arrendadas do Centro de Ciências Tecnológicas da Terra e do Mar da Universidade do Vale do Itajaí.

# ABSTRACT

Telemetry is the technique that measure quantities, transmiting the results to a control center responsible for the interpretation, presentation and/or storage of these measurements. The modes of data transmition can vary, ranging from conventional telephone lines to the Internet. How the transactions must occur has been standardized by numerous organizations, with the objective of turning fully compliant the industry that manufactures telemetry equipments. With the new technologies, there is room for creating more standards for telemetry transactions in software application layer. This work fullfills this need by proposing a protocol for telemetry data transactions using the technology of Web Services. This standardization has immediate application in the system for tracking vessels of the Centro de Ciências Tecnológicas da Terra e do Mar of the Universidade do Vale do Itajaí.

# I - INTRODUÇÃO

## 1 APRESENTAÇÃO

Com o desenvolvimento da tecnologia e de sua portabilidade para sistemas embarcados portáteis, ou mesmo para o monitoramento constante, tem-se observado uma crescente necessidade da comunicação entre sensores e sistemas de armazenamento e análise dos dados coletados. Para que se possa efetuar essa coleta remotamente e muitas vezes em tempo real, são utilizadas técnicas de telemetria, que é o processo pelos quais parâmetros de um objeto são medidos e os resultados transmitidos para uma estação distante, onde então serão armazenados, apresentados e/ou analisados (L-3 COMMUNICATIONS, 2003).

O processo da telemetria envolve agrupar medidas (tais como pressão, velocidade e temperatura) em uma estrutura que possa ser transmitida em um único fluxo de dados. Uma vez recebido, o fluxo de dados é separado nos componentes medidos originalmente para que possa então ser analisado (ibidem).

Entre os exemplos de aplicação de telemetria pode ser citado o uso em aeronaves, monitorando o posicionamento e rumo de viagem, bem como o estado de funcionamento de instrumentos ou outras peças de um sistema embarcado (NASA, 2003). Além disso, o uso de telemetria também está presente no desenvolvimento dessas aeronaves, onde, durante os testes iniciais de vôo, uma aeronave executa testes de manobras. Os dados críticos de uma manobra são transmitidos para os engenheiros em uma estação em terra, onde os resultados são vistos em tempo real ou analisados dentro de segundos durante a manobra. Após a análise em tempo real, a próxima manobra poderá ser executada ou o piloto poderá receber instruções para retornar a base por razões de segurança (L-3 COMMUNICATIONS, 2003).

Um outro exemplo de uso da telemetria é a monitoração contínua dos níveis dos rios, em locais estratégicos, nas principais bacias hidrográficas brasileiras. O objetivo é gerenciar afluências e defluências dos reservatórios de usinas hidrelétricas e minimizar os efeitos de situações hidrológicas críticas (cheias e estiagens) por meio da disponibilização e emissão de boletins hidrológicos para o acompanhamento da evolução dos níveis dos rios (ANEEL, 2003).

É importante ressaltar o sistema RASTRO, que foi utilizado como estudo de caso, sendo o primeiro sistema a implementar o padrão proposto. O RASTRO é um sistema acessado via *web* que apresenta as informações na forma de mapas de navegação contendo as últimas posições e os dados traçados de uma ou várias embarcações. Por trás desse sistema, encontra-se um módulo que é responsável por receber os dados de empresas de rastreamento, que acompanham via satélite a localização das embarcações na costa brasileira. O capítulo 5 aborda de forma mais completa o sistema RASTRO.

Em um sistema de telemetria, a aquisição dos dados inicia-se quando os sensores medem uma quantidade de atributos físicos e transformam essas medidas em uma unidade de valor planejado. Esses sensores estão ligados a um condicionador de sinal, que fornece energia para que eles funcionem ou modifiquem o tipo do sinal para o próximo estágio da aquisição. Um multiplexador é utilizado para capturar os valores medidos em série de forma análoga e emití-los para uma saída (*output*) como um único fluxo de dados (L-3 COMMUNICATIONS, 2003).

De nada adiantaria coletar dados se não fosse possível armazená-los e tratá-los de forma que os tornassem informações úteis para alguma aplicação. Dessa forma, poder-se-ia propôr uma solução em que esses dados coletados fossem transmitidos utilizando uma tecnologia bastante inovadora para troca de informações: os *Web Services*.

A idéia dos *Web Services* é criar componentes de *software* que podem ser ativados a distância via Internet, trocando informações pela rede na forma de arquivos no padrão XML (*Extensible Markup Language*). Cada componente se autodescreve, o que facilita o seu uso, e eles podem ser listados em diretórios para que sejam encontrados mais facilmente na Internet (GREGO, 2002).

## 2 JUSTIFICATIVA

Em Ciência da Computação são contempladas diversas áreas de conhecimento, entre elas: Sistemas Operacionais, Redes de Computadores, Programação e Modelagem de Sistemas. Para o desenvolvimento desta proposta, que aborda a comunicação entre sistemas de forma automatizada, todas estas áreas foram colocadas em prática.

Como exemplo de aplicação da proposta em uma sub-área de sistemas de telemetria - o rastreamento de veículos tem sido explorado por diversas empresas que utilizam a transferência de dados de telemetria por meio de satélite. Essas empresas fornecem, além do equipamento utilizado para a coleta e transmissão dos dados, um sistema que controla os dados coletados, de forma a apresentá-los e tratá-los no centro de controle. Isso torna-se uma grande desvantagem para o cliente, pois uma vez adquirido esse sistema, o cliente estará preso a ele, não podendo personalizar a aplicação existente, ou mesmo substituí-la por uma aplicação melhor. Dessa forma, ao padronizar a comunicação entre a empresa de rastreamento e o cliente, permite-se a escolha de qualquer sistema para o tratamento dos dados, independente da empresa de rastreamento que será utilizada, ou mesmo o desenvolvimento interno de uma aplicação que supra as necessidades da empresa.

## 3 IMPORTÂNCIA

Muitas vezes é caro ou inviável o uso de cabos ou equipamentos que possam transmitir dados coletados por um sensor para um computador que efetue o processamento e armazenamento dos dados. O uso de um *Web Service* que possua características apropriadas para o recebimento desses dados apresenta-se como uma alternativa de baixo custo, pois utiliza a infra-estrutura existente da Internet para a transmissão dos dados.

Atualmente, são vários os sistemas de telemetria disponíveis no mercado, porém, esses sistemas possuem um protocolo proprietário ou fechado para a troca de informações, como é o caso do Padrão NMEA (*National Marine Electronics Association*), que serve para a troca de informações entre equipamentos marítimos (NMEA, 2003). Esse projeto aborda de forma diferente a troca de informações, utilizando, para isso, a tecnologia de *Web Service*. Esse modelo foi incorporado ao à um sistema de monitoramento de embarcações, mostrando, dessa forma, que além de inovador, é aplicável a sistemas já existentes.

Também é importante ressaltar que essa troca de informações é efetuada de forma segura, o que significa que esse modelo pode ser utilizado na esfera comercial. Para assegurar a troca dos dados, foi utilizado o protocolo HTTPS (*Hyper Text Transfer Protocol Secure*), que apresenta o uso de HTTP (*Hyper Text Transfer Protocol*) encapsulado sobre SSL (*Secure Socket Layer*) permitindo o envio e recebimento de mensagens pela Internet de forma criptografada. Também foi utilizado

uma autenticação do usuário no momento de execução do método, permitindo o acesso ao *Web Service* somente a sistemas autorizados.

# 4 OBJETIVOS DO TRABALHO

## 4.1 Objetivo Geral

O objetivo geral deste trabalho consiste na proposta de um padrão de *Web Service* que atenda as necessidades de troca de informações em sistemas de telemetria. Especificamente, este padrão atende a aplicação em sistemas de monitoramento.

## 4.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

?? Avaliar o estado da arte quanto ao emprego de *Web Services* para a troca de informações entre sistemas, quanto a integridade e consistência dos dados e quanto a segurança;

?? Definir o Padrão de *Web Service* a ser utilizado em consonância com o estado da arte;

?? Implementar e validar o Padrão de *Web Service*;

# 5 METODOLOGIA

A metodologia seguida no desenvolvimento do presente projeto baseia-se nos seguintes tópicos:

1. Levantamento bibliográfico sobre telemetria;
2. Estudo da linguagem de marcação XML (*Extensible Markup Language*);
3. Levantamento bibliográfico sobre *Web Services*;
4. Pesquisa de formas para a transmissão dos dados no nível de aplicação;
5. Analise e comparação as formas de transmissão de dados no nível de aplicação;
6. Pesquisa das formas de segurança disponíveis pelos protocolos usados para assegurar a transferência dos dados;

7. Modelagem do sistema de *Web Service* utilizando UML (*Unified Modeling Language*);
8. Implementação de um *Web Service* complementando o Sistema RASTRO;
9. Validação do sistema;
10. Documentação.

O **levantamento bibliográfico sobre telemetria**, aborda o que é a telemetria, e apresenta duas de suas classificações (telemetria via Internet e via rádio), também descreve algumas de suas vantagens e desvantagens. A finalidade dessa etapa é situar o presente trabalho na área de telemetria.

Na etapa de **estudo da linguagem de marcação XML**, é apresentado, de forma sucinta, o que é o XML, o *namespace* e o XML *Schema*. Também são apresentados os tipos proporcionados pelo XML *Schema*, e onde o XML se encaixa na transferência de informações.

O **levantamento bibliográfico sobre *Web Service*** englobou a pesquisa em livros e *sites* referente a *Web Service*, XML, XML-RPC, SOAP e redes de computadores. Os temas apresentados nessa seção abrangem o UDDI (*Universal Description, Discovery and Integration*) e sua função; o documento WSDL (*Web Services Description Language*), sua estrutura e suas formas de operação; e algumas vantagens do uso de *Web Services*.

Em **pesquisa de formas para a transmissão de dados no nível de aplicação**, são abordados os protocolos disponíveis para o desenvolvimento de um *Web Service* (SOAP e XML-RPC), sua arquitetura, tipos suportados, formato do arquivo XML da requisição e resposta e os elementos e atributos suportados por cada protocolo.

Na **análise e comparação das formas de transmissão de dados no nível de aplicação**, é apresentado um breve comparativo entre o XML-RPC e o SOAP, e uma tabela com algumas características e seus possíveis suportes.

No tópico **pesquisa das formas de segurança existentes nos protocolos usados para assegurar a transferência dos dados**, são estudados os meios de garantir a segurança na troca de informações entre o servidor e o cliente. Entre os assuntos abordados, estão a autenticação, não repudiação, integridade e privacidade.

Foi elaborado em **modelagem do sistema de *Web Service* utilizando UML**, o modelo de casos de uso e de entidade e relacionamento do *Web Service*. Foi escolhida a modelagem em UML, pois essa notação vem emergindo como um padrão para a modelagem de sistemas.

O tópico **implementação do *Web Service*** pode ser considerado o mais importante do trabalho, pois nele foram materializados os conceitos estudados. Inicialmente foi elaborado um protótipo utilizando todo o padrão proposto, e, em seguida, foi implementado o caso prático adotado: o sistema RASTRO.

A **Validação do sistema** foi efetuada com a substituição do método empregado inicialmente no sistema RASTRO para a recepção de dados, pelo padrão de *Web Service* proposto por este trabalho.

A **Documentação** está presente neste trabalho em forma de anexo (Anexo IV). Ela visa facilitar o desenvolvimento de novas ferramentas baseadas em *Web Service* voltadas para sistemas de telemetria.

# II - REVISÃO BIBLIOGRÁFICA

## 1 TELEMETRIA

### 1.1 Introdução

Telemetria é a técnica que mede quantidades, transmitindo os resultados para um centro de controle distante que é responsável por interpretar, apresentar e/ou armazenar essas quantidades medidas. Os meios de transmissão de dados utilizados são muito variados, podendo ser utilizadas, por exemplo, desde linhas telefônicas convencionais até a transmissão por ondas de rádio (L-3 COMMUNICATIONS, 2003).

A telemetria permite a coleta de dados em locais de difícil acesso ou perigosos para os seres humanos. Esses pontos de coleta podem ser próximos dos centros de controles ou como normalmente aplicados, podem estar a centenas de quilômetros (IN4MA REMOTE MONITORING, 2003).

Nas atuais aplicações de telemetria, que possuem um extenso número de valores que podem ser medidos, é impraticável e muitas vezes caro, o uso de canais distintos para a transmissão contínua de cada valor mensurado. Dessa forma, o processo de telemetria envolve o agrupamento dos valores coletados (como o posicionamento, a temperatura e o fluxo) dentro de uma estrutura que possa ser transmitida em um único fluxo de dados. Uma vez recebido esse fluxo no centro de controle, a estrutura é decomposta nos valores coletados inicialmente que então serão tratados, armazenados ou visualizados (L-3 COMMUNICATIONS, 2003).

Os sistemas de telemetria podem ser caracterizados de várias formas. Como exemplo serão citados dois modelos: a telemetria sem fio e a telemetria via Internet, descritas a seguir.

A **telemetria sem fio** pode ser definida como sendo uma transferência automática de dados entre duas máquinas por meio de uma rede sem fio com o intuito de monitoramento e controle (ERICSSON, 2003). Para aplicações de telemetria sem fio são utilizadas desde antenas de rádio até a transmissão de dados por meio de satélites.

A grande maioria dos sistemas de telemetria sem fio utiliza-se dos sinais VHF (*Very High Frequency*) ou UHF (*Ultra High Frequency*) para a transferência de dados. A freqüência VHF estende-se de 30 MHz (*Mega Hertz*) até 300 MHz, enquanto a freqüência UHF cobre até 3000 MHz. As permissões para o uso dessas freqüências devem ser dadas previamente pela ANATEL (Agência Nacional de Telecomunicações) evitando interferências causadas por outros usuários (DAVE, 2003). De acordo com o mesmo autor, existem sistemas de telemetria sem fio que atuam pela emissão de sinais dos centros de controle, onde os *modems* de rádio operam de forma transparente, simplesmente provendo os meios para comunicação entre o centro de controle e os pontos remotos. Em outros casos, os *modems* de rádio nos pontos remotos possuem capacidade de efetuar o envio de sinais por si só. Dessa forma, eles podem rapidamente criar uma rede em estrela onde o centro de controle situa-se ao centro da rede. O processo funciona da seguinte forma:

?? O centro de controle envia um sinal para a estação remota, perguntando se ela possui algum dado para ser enviado;

?? A estação remota responde enviando a estrutura com os dados, ou apenas um sinal indicando que não possui nada a repassar;

?? O centro de controle então entra em contato com a próxima estação remota repetindo o processo de coleta de dados;

?? Após contatar todas as estações remotas, o centro de controle inicia o processo novamente.

Em outros casos, a rede utilizada entre as estações remotas e o centro de controle assemelha-se ao método de acesso de uma rede de computadores, o CSMA (*Carrier Sense Multiple Access*), que trabalha mais rapidamente que o método de *pooling*. É uma rede unida, onde todas as estações ficam aguardando serem contatadas e apenas a estação que for solicitada responderá. Caso duas estações tentem transmitir no mesmo momento não será possível, assim ambas param de transmitir e após um período aleatório de tempo tentam novamente. A primeira estação que conseguir iniciar a transmissão o fará até o fim, caso outra antena tente efetuar a transmissão receberá um sinal de "ocupado". Após a liberação do sinal pela primeira estação, a segunda poderá transmitir seus dados.

Segundo Dave (2003), as vantagens no uso de sistemas de telemetria sem fio são:

?? Opera em uma grande variedade de condições;

?? Responde rapidamente;

?? É possível a comunicação com locais onde não podem ser utilizadas linhas telefônicas;

?? Pode ser reinstalado em outro local com facilidade;

?? Permite comunicação móvel.

E, conforme o mesmo autor, as desvantagens são:

?? Ondas de rádio viajam no espaço em um plano horizontal. A curvatura da Terra restringe o alcance entre dois pontos em 40 quilômetros. Dessa forma, torna-se necessário um repetidor que esteja localizado no ponto mais alto entre o local de coleta de dados (sensor) e o Centro de Coletas, permitindo que o sinal possa ser retransmitido.

?? A antena é parte vital do sistema sem fio. O projeto da antena deverá considerar obstáculos entre os dois pontos (morros, edifícios, etc), o tipo de solo, a velocidade do vento, entre outros. Deverá ser considerado, dessa forma, o custo de retransmissores entre os dois pontos.

?? Torres de alta tensão, transformadores e grandes equipamentos elétricos geram impulsos eletro-magnéticos que distorcem os sinais de rádio. Porém, existem *modems* de rádio que detectam esses impulsos podendo repetir os sinais perdidos, resolvendo esse problema.

Por outro lado, a **telemetria via Internet** descreve a utilização da infra-estrutura da Internet (ou, mais precisamente, as comunicações baseadas em IP (*Internet Protocol*)) como sendo uma conexão significativa entre um sensor de coleta de dados e o centro de coletas (SRC, 2003).

O sistema de telemetria via Internet funciona da mesma forma que uma aplicação cliente/servidor. A estação remota está conectada a um provedor de acesso à Internet. Da mesma forma, do outro lado, o centro de coletas também está conectado a um provedor de acesso à

Internet. Para a transferência de dados entre a estação remota e o centro de coletas, são utilizados protocolos baseados em IP (SRC, 2003).

O uso da Internet para o transporte de dados por longas distâncias reduz o custo de operação de um sistema remoto de monitoração e controle (telemetria), substituindo as chamadas telefônicas de longas distâncias (ou linhas dedicadas) pela assinatura de um provedor de Internet. Pode-se destacar ainda as seguintes vantagens no uso desse tipo de telemetria (ibidem):

?? O uso da Internet para a transferência de dados fornece um meio de comunicação mais confiável e com redundância de conexão a longas distâncias;

?? A pilha de protocolos TCP/IP é muito confiável e muito bem testada, provendo base para outros protocolos de alto nível como o FTP (*File Transfer Protocol*), o HTTP (*HyperText Transfer Protocol*) e o SMTP (*Simple Mail Transport Protocol*);

?? Várias sessões de comunicação podem ser simultaneamente manipuladas em uma conexão baseada em IP;

?? Devido à popularização da Internet, os custos referentes ao acesso estão se tornando cada vez menores.

Os sistemas de telemetria atuais são construídos a partir de produtos comerciais de prateleira. Eles possuem muitos elementos em comum, os quais podem ser configurados de modo a se adequar às necessidades específicas de cada aplicação (L-3 COMMUNICATIONS, 2003).

## 1.2 Considerações

Esta capítulo teve como intuito apresentar os conceitos que cercam a telemetria, exemplos de sua classificação e algumas vantagens e desvantagens. O objetivo é apenas apresentar de forma conceitual, uma vez que o presente projeto não visa desenvolver uma aplicação que atenda diretamente a comunicação existente hoje na telemetria, e sim apresentar os benefícios de uso da tecnologia de *Web Services* para o transporte de dados telemétricos.

Baseando-se nesse objetivo é de suma importância definir e descrever um padrão que atenda todos os requisitos no que se trata da transmissão de dados telemétricos, uma vez que o documento

final poderá ser utilizado por diversas empresas ou entidades que visam tornar seus sistemas ou serviços o mais interoperável possível.

# 2 PADRÃO

## 2.1 Introdução

A padronização é o esforço para se alcançar soluções comuns, sendo elaborada em cooperação entre representantes de indústrias, usuários, comerciantes e consumidores. O propósito da padronização é criar uma plataforma de trabalho comum de onde, por exemplo, todos saibam o significado de conceitos diferentes tendo um procedimento uniforme ou tendo um mínimo de requisitos definidos. O esforço pela padronização é freqüentemente dirigida por comitês técnicos, grupos de trabalhos ou projetos onde seus participantes são oriundos de empresas, organizações ou autoridades interessadas (OMNITOR, 2003).

A padronização é um assunto internacional. Especialistas estão participando ativamente em padronizações internacionais com intuito de repassar pontos de vista das suas organizações e criar padrões técnicos que irão suportar o desenvolvimento da comunicação no futuro (ibidem). Qualquer padrão é um trabalho coletivo. Comitês de fabricantes, usuários, organizações de pesquisa, departamentos do governo e consumidores trabalham juntos para descrever padrões que se encontram com as demandas da sociedade e da tecnologia (BRITISH STANDARDS, 2003).

## 2.2 Definição de Padrão

O padrão é definido como um documento, estabelecido pelo consenso e aprovação de uma entidade reconhecida como por exemplo a ISO (*International Organization for Standardization*) ou a IEC (*International Electrotechnical Comission*), que fornece para uso comum ou repetitivo regras, diretrizes ou características para atividades ou seus resultados (BRITISH STANDARDS, 2003).

Mas os padrões não são apenas isso. Eles podem existir para objetos (*e.g.* lâmpadas), e progressivamente fazer as coisas acontecer (*e.g.* serviços), mas principalmente os padrões representam um nível indispensável de experiência em qualquer área determinada. No contexto de

contratos públicos ou comércio internacional, os padrões são essenciais para simplificar e esclarecer relações contratuais (BRITISH STANDARDS, 2003).

Os padrões facilitam o mundo comercial removendo barreiras técnicas para o comércio, levando a novos mercados e ao crescimento econômico. Os padrões eletro-técnicos são conciliados internacionalmente pela ISO e pela IEC (IEC, 2003).

## 2.3 A importância dos Padrões

Os padrões representam uma importante função na vida cotidiana. Eles estabelecem a proporção, capacidade e modelo de um produto, processo ou sistema, e especificam o desempenho de produtos ou pessoal. Eles também podem definir termos de forma que não exista mal-entendido entre aqueles que utilizam o padrão (ANSI, 2003).

Por exemplo, compradores e usuários de produtos são avisados com antecedência quando um produto é de má qualidade, perigoso, inadequado ou incompatível com outro equipamento que já possui. Quando os produtos se adequam às expectativas do comprador, a tendência é escolhê-los. Freqüentemente, os compradores não tem consciência da função representada pelos padrões no aumento de níveis de qualidade, segurança, confiança e eficiência - também fornecendo esses benefícios a um custo econômico (ISO, 2003a).

## 2.4 Normalização

A normalização é a atividade que estabelece, em relação a problemas existentes ou potenciais, prescrições destinadas à utilização comum e repetitiva com vistas à obtenção do grau ótimo de ordem em um dado contexto (ABNT, 2003a).

A normalização é a busca de um padrão o mais otimizado possível, portanto pode-se inferir que a normalização é a maneira de organizar as atividades pela elaboração, publicação e promoção do emprego de Normas e Regras, visando contribuir para o desenvolvimento econômico e social do País. Ou seja, estabelece soluções para problemas de caráter repetitivo existentes ou potenciais (SEBRAE SC, 2003).

## 2.5 Entidades de Normalização

Existem várias entidades responsáveis pela definição de padrões no mundo. Praticamente, cada país possui uma entidade responsável pela definição dos padrões e normas nacionais. Também existem órgãos responsáveis por padrões regionais e internacionais. Abaixo serão relacionados alguns órgãos nos três âmbitos (nacional, regional e internacional).

### 2.5.1 ISO (International Organization for Standardization)

A ISO é uma organização não governamental que foi estabelecida no ano de 1947, sendo formada por institutos nacionais de padrões localizados em 147 países. O secretariado geral está localizado na cidade de Genebra (Suíça), de onde todo o corpo é coordenado. Ele está apto a atuar como uma organização "ponte", na qual um consenso pode ser alcançado em soluções que adequam os requisitos do negócio e as vastas necessidades da sociedade (ISO, 2003a).

### 2.5.2 ABNT (Associação Brasileira de Normas Técnicas)

A ABNT foi fundada no ano de 1940 e é o órgão responsável pela normalização técnica do Brasil, fornecendo a base necessária para o desenvolvimento tecnológico brasileiro. Ela é uma entidade privada e sem fins lucrativos reconhecida como único Foro Nacional de Normalização. A ABNT é representante nacional da ISO e do Comitê Mercosul de Normalização (CMN), tendo um papel triplo no que refere-se ao desenvolvimento da normalização (ABNT, 2003b).

### 2.5.3 CMN (Comitê Mercosul de Normalização)

O objetivo do CMN é criar normas voluntárias para os produtos e serviços que circulam no bloco do Mercosul. Instituída no ano de 1991, a CMN tem sede na cidade de São Paulo de onde são coordenados os trabalhos de 19 comitês setoriais de normalização. Atualmente, são disponibilizados cerca de 200 normas e 700 projetos de trabalho pela CMN (AMN, 2003).

### 2.5.4 IEC (International Electrotechnical Comission)

Fundado oficialmente em 1906, em Londres (Inglaterra), o IEC é a principal organização global responsável pela preparação e publicação de padrões internacionais que envolva toda a tecnologia elétrica, eletrônica ou relacionada. A patente da IEC engloba toda a eletro-tecnologia incluindo eletrônicos, magnéticos e eletromagnéticos, eletroacústicos, multimídia, telecomunicação e produção e distribuição de energia, assim como técnicas gerais associadas como a terminologia e símbolos, a compatibilidade eletromagnética, dimensão e desempenho, fidelidade, *design* e desenvolvimento, segurança e também o ambiente. Para se ter uma idéia dos trabalhos realizados pela IEC, pode-se citar que ela foi responsável pelo estabelecimento de várias unidades elétricas, entre elas o Hertz como unidade de freqüência, o Gauss como unidade de densidade de fluxo magnético e o Maxwell como unidade de fluxo magnético (IEC, 2003).

### 2.5.5 NMEA (National Marine Electronics Association)

A NMEA tornou-se uma desenvolvedora, mantenedora e distribuidora de vários documentos e padrões relacionados a indústria eletrônica naval. O padrão NMEA foi aceito como forma comum de distribuição específica de dados entre instrumentos navais, equipamentos de navegação e comunicação quando interconectados por meio de uma *interface* apropriada. Esse padrão é adotado por entidades concorrentes ou companhias desde meados de 1970 (NMEA, 2003).

### 2.5.6 W3C (World Wide Web Consortium)

A W3C foi criada em 1994 para conduzir a WWW (*World Wide Web*) ao seu completo potencial, desenvolvendo protocolos comuns, promovendo a sua evolução e assegurando a sua interoperabilidade. A W3C possui cerca de 400 organizações membro por todo o mundo e ganhou o reconhecimento internacional pelas suas contribuições no crescimento da *web*. Sua missão é promover a interoperabilidade e incentivar um fórum aberto para discussão, onde os comitês da W3C conduzem a evolução técnica da *web*. Em apenas 7 anos a W3C desenvolveu mais de cinqüenta especificações técnicas para a infra-estrutura da *web* (JACOBS, 2000).

## 2.6 Exemplos de Padrões

A NBR ISO/IEC 17799 (Tecnologia da informação - Código de prática para a gestão da segurança da informação) é um conjunto de recomendações para a gestão de segurança da informação que deve ser utilizada pelas pessoas responsáveis em implantar ou administrar a segurança em uma organização. Seu propósito é fornecer uma base comum no desenvolvimento de normas de segurança organizacional e das práticas efetivas da gestão de segurança, fornecendo confiança nos relacionamentos entre as organizações. A ISO 17799 atua em segurança da informação considerando tecnologia, processos e pessoas (NERY & PARANHOS, 2003).

A NBR 6023 (Informação e Documentação: Referências - Elaboração) é uma das Normas Brasileiras mais utilizadas, sendo responsável por especificar os elementos que deverão ser inclusos nas referências bibliográficas, fixando a ordem dos elementos da referência e estabelecendo convenções para a transcrição e apresentação da informação original do documento e/ou outras fontes de informação. Ela orienta a preparação e compilação de referências de material utilizado para produção de documentos e para inclusão de bibliografias, resumos, resenhas e outros (ABNT, 2000).

## 2.7 Construção de Padrões

Grande parte das especificações ou padrões são criadas a partir de comitês técnicos ao invés de um único indivíduo. Por exemplo, os grupos de Trabalho da W3C tipicamente consistem de 10 a 20 pessoas, que trabalham juntos em uma nova tecnologia por um ou mais anos (BOS, 2003).

A criação de padrões a partir de comitês pode gerar especificações que possuem remendos a partir de soluções inconsistentes e muitas vezes redundantes, além de difícil compreensão, mas na realidade não produz automaticamente resultados ruins. Mais pessoas desenvolvendo uma especificação significa mais checagem de erros, mais criatividade em encontrar soluções para os problemas e mais experiência em saber o que foi trabalhado ou não no passado. Mesmo assim, o problema de comitês ainda deve ser evitado. Um número recomendado é de no máximo 15 pessoas, pois grupos maiores tendem a dar forma a subgrupos informais perdendo muito tempo na comunicação ao invés do desenvolvimento (ibidem).

Por outro lado, grupos menores produzem especificações mais consistentes e mais fácil de serem utilizadas, mas podem omitir algumas características desconhecidas pelo grupo e mas necessárias para outras pessoas. A solução parece ser a criação de um amplo círculo de pessoas interessadas sobre o assunto, na forma de uma lista de discussão pública (BOS, 2003).

É dessa forma que a W3C desenvolve suas tecnologias: é criado um grupo de trabalho composto por *experts* no assunto e uma lista de discussão pública para as pessoas interessadas, pois podem existir pessoas interessadas um único detalhe de uma especificação ou no desenvolvimento do padrão como um todo. Já a IETF (*Internet Engineering Task Force*) mostra que é possível desenvolver tecnologias com um único grupo aberto de pessoas sem a distinção entre as pessoas comprometidas com o projeto e as que possuem um mínimo de esforço. A diferença é que a IETF trabalha com especificações de mais baixo nível, enquanto a W3C, por estar mais próxima do usuário, possui mais interessados.

No programa de especificação da OGC (*Open Gis Consortium*) o comitê técnico e o comitê de planejamento trabalham em um processo de consenso formal para chegar a uma especificação Open Gis aprovada (ou "adotada"). O comitê técnico da OGC utiliza um documento intitulado "*Technical Committee Policies and Procedures*" para cumprir o desenvolvimento de uma especificação. Este documento esboça as políticas e procedimentos utilizados pelo comitê técnico do Open Gis Consortium para o desenvolvimento de uma especificação. Essas políticas podem ser alteradas por votação do comitê técnico ou comitê de administração do OGC (OPEN GIS Consortium, 2003).

Segundo a ISO (2003b), o processo de desenvolvimento de um padrão internacional nos moldes da ISO é o resultado de um acordo entre o corpo de membros. O Padrão pode ser utilizado como foi apresentado ou pode ser implementado através da junção de padrões nacionais de países diferentes. Um padrão internacional é desenvolvido por um comitê e subcomitês técnicos da ISO seguindo seis passos:

?? Estágio 1: Estágio de Proposta

?? Estágio 2: Estágio Preparatório

?? Estágio 3: Estágio de Comitê

?? Estágio 4: Estágio de Investigação

?? Estágio 5: Estágio de Aprovação

?? Estágio 6: Estágio de Publicação.

Se um documento com um determinado grau de maturidade estiver disponível no início da padronização de um projeto, por exemplo um padrão desenvolvido por outra organização, é possível que ele omita alguns estágios iniciais. No então chamado "*Fast-Track Procedure*" um documento é submetido diretamente para a aprovação como um DIS (*Draft International Standard*) ao corpo de membros da ISO (Estágio 4) (ISO, 2003b).

## 2.8 Considerações

Em Ciência da Computação, o termo "padrão" é amplamente utilizado em suas diversas áreas, como em padrões de *software* (engenharia de *software*), padrões de protocolo (redes de computador), padrões de linguagem (compiladores), padrões de protocolos e concorrência (sistemas operacionais).

Neste trabalho, o termo padrão tem como referência uma proposta de especificação de linguagem de comunicação entre servidores e clientes que necessitam trocar informações de telemetria. Este padrão é definido através da linguagem WSDL (Seção 3.5) utilizada para descrever *Web Services*.

# 3 WEB SERVICES

## 3.1 Introdução

Várias organizações têm definido *Web Service* de formas diferentes, mas em sua essência *Web Service* é um componente de *software* que interage com outro componente de *software* distinto através de protocolos de rede conhecidos. Com o uso de XML, o acesso aos serviços tornou-se independente das suas implementações. Assim, um *Web Service* oferece tanto o benefício do componente de *software* quanto do uso da *Web* para a transferência dos dados (AYALA et al. 2002).

Os componentes representam uma funcionalidade implementada em uma "caixa-preta", a qual pode ser reutilizada sem a preocupação de como o serviço foi implementado. As aplicações acessam os *Web Services* através de protocolos padrão, como HTTP e SOAP (*Simple Object Access Protocol*) e transferem os dados no formato XML, diferentemente das tecnologias anteriores que utilizam protocolos específicos (DCOM - *Distributed Component Object Model; RMI - Remote Method Invocation*; ou IIOP - *Internet Inter-Operability Protocol*) (OPV, 2003).

Em um curto espaço de tempo, as corporações têm observado o surgimento dos *Web Services* para uma grande variedade de aplicações, como o gerenciamento de armazenamentos e a gerência de relacionamento com o cliente, e muitas outras áreas específicas, como o monitoramento de preços das ações e os leilões virtuais. As atuais tendências das corporações estão acelerando a criação e disponibilidade desses serviços (AYALA et al., 2002).

Conforme apresentado na Figura 1, o *Web Service* pode se registrar em um repositório central, onde *softwares* clientes que necessitam dos serviços oferecidos pelo *Web Service* poderão pesquisá-lo. Uma vez localizado, o *software* cliente receberá uma referência para o serviço encontrado, podendo usufruir dos serviços apenas executando os vários métodos implementados com a ajuda das *interfaces* públicas. Assim, cada serviço precisa publicar sua *interface* para que os *softwares* clientes a utilizem (ibidem).

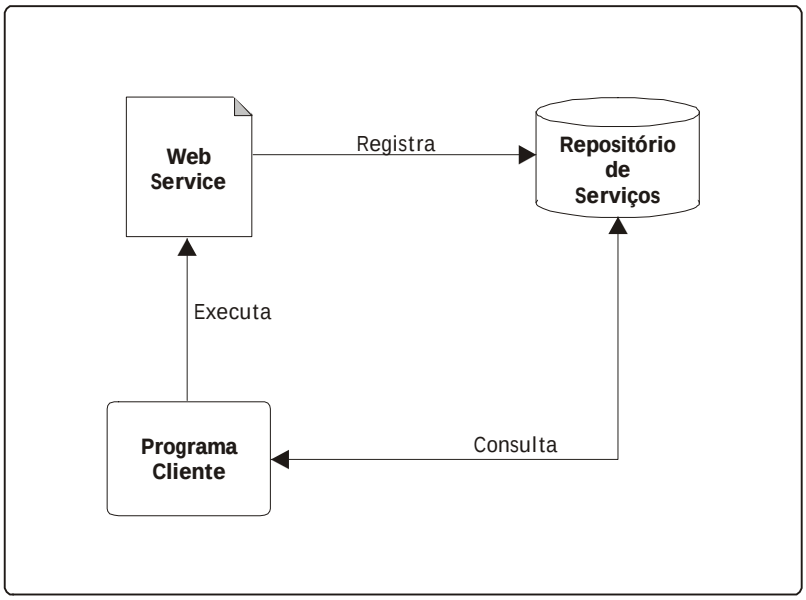



Figura 1: Funcionamento Simplificado de um *Web Service*.
Fonte: Ayala et al. (2002).

Austin et al. (2002) define um *Web Service* como sendo um sistema identificado por um URI (*Uniform Resource Identifiers*), que publica uma *interface* e um meio de comunicação lógico que são definidos e descritos usando XML. Essa definição pode ser descoberta por outros sistemas, que então podem interagir com o *Web Service* conforme descrito pela sua definição, usando mensagens baseadas em XML que são transportadas sob protocolos de Internet.

## 3.2 Vantagens da Utilização de Web Services

Segundo Ayala et al. (2002), são inúmeras as vantagens do uso de *Web Services*, sendo as mais importantes: interoperabilidade, integração, eficiência de implementação, reciclagem de código e modularidade.

?? **Interoperabilidade**: É a capacidade de dois ou mais sistemas interagirem entre si trocando dados conforme um método proposto, nesse caso os *Web Services*. Não importa qual o segmento de atuação de uma empresa, se for preciso permitir a outros sistemas externos o acesso a documentos de negócios compartilhados, a interoperabilidade faz-se necessária.

?? **Integração**: Os *Web Services* são ótimos integradores de aplicações, permitindo unir programas separados dentro de uma única aplicação composta. Em um cenário real, se uma organização possui uma grande variedade de aplicações ou programas personalizados através de vários departamentos que não trocam informações entre si, é possível expor a funcionalidade e os dados de cada aplicação existente utilizando essa tecnologia, de forma a criar uma aplicação composta garantindo a interoperação entre essas aplicações.

?? **Eficiência de Implementação**: Fazendo com que as aplicações tornem-se interoperáveis entre si, não será mais necessário investir totalmente em um novo sistema para a empresa. Apenas será necessário ajustar as aplicações existentes para torná-las compatíveis com *Web Services*. Eles também são caminhos eficientes para expor códigos legados a disponibilizar um novo serviço para consumidores externos.

?? **Reciclagem de Código**: Os *Web Services* são desenvolvidos a partir de componentes reutilizáveis. Dessa forma, se a aplicação existente é baseada em componentes, a migração se tornará mais fácil.

?? **Modularidade**: *Web Services* também garantem a modularidade ajudando na construção de aplicações para empresas que necessitam disponibilizar uma *interface* externa aos consumidores, promovendo uma arquitetura *plug-and-play*.

## 3.3 XML

O XML (*Extensible Markup Language*) descreve uma classe de objetos de dados chamado documento XML responsável por delinear parcialmente o comportamento de programas de computador que os processa. O XML é um perfil de aplicação ou forma restrita do SGML (*Standard Generalized Markup Language*) (BRAY et al., 2000).

Documentos XML são formados de unidades de armazenamento chamadas entidades, que contém dados analisados ou não analisados. Os dados analisados são compostos de caracteres, sendo que alguns se caracterizam apenas como simples caracteres, enquanto outros formam a marcação. A marcação codifica a descrição do *layout* do documento armazenado e a estrutura lógica. O XML fornece um mecanismo para impôr limitações no *layout* armazenado e na estrutura lógica (ibidem).

A diferença entre o HTML (*HyperText Markup Language*) e o XML é que as *tags* do HTML servem para a apresentação de dados, enquanto as *tags* do XML têm por finalidade descrever os dados. Por exemplo, um documento XML descreve o dado como o nome do empregado, o sexo e a idade. O dado é descrito pela criação de *tags* compreensíveis facilmente, sendo que essas *tags* auxiliam na criação de documentos bem formatados. O XML é eficaz na troca de dados estruturados entre aplicações desde que os documentos XML possuam uma estrutura definida que a qualifique como um documento bem formatado (AYALA et al., 2002).

### 3.3.1 Namespaces

Segundo Bray, Hollander & Layman (1999), um XML *namespace* é uma coleção de nomes identificados por uma referência URI, que são utilizados em documentos XML como tipo dos elementos ou nome dos atributos. A maioria dos nomes de *namespaces* são escritos usando o formato de uma URL, utilizando o protocolo HTTP. Por exemplo: `xmlsns="http://www.w3c.org/myschema"`.

Apesar da declaração de *namespace* apontar para uma determinada localização, o principal objetivo em utilizar-se de URIs ao criá-los é o de disponibilizar um nome único para o *namespace* que possa ser associado a uma determinada empresa. Os *namespaces* são sensíveis a “caixa”, ou seja, palavras com letras maiúsculas diferem de letras minúsculas (TESCH, 2002).

### 3.3.2 XML Schema

Segundo Thompson et al. (2001), um XML *Schema* consiste de componentes tais como definição de tipos e declaração de elementos, que podem ser utilizados para avaliar itens da informação do atributo e a validade de elementos bem formatados, além disso, é possível especificar acréscimos para esses itens e seus descendentes. Esses acréscimos criam informações explícitas que talvez tenham sido implícitas no documento original, tais como valores normalizados e/ou valores padrões para atributos e elementos, os tipos de elementos e itens da informação do atributo. Tesch (2002) acrescenta que o XML *Schema* utiliza *namespaces* para limitar os nomes dos elementos e atributos.

Para a criação de documentos SOAP e WSDL, faz-se necessário o conhecimento referente ao XML *Schema* para a especificação dos tipos de dados (TIDWELL, 2001).

#### 3.3.2.1 Tipos de Dados Simples e Complexos

Em um XML *Schema* todos os tipos de dados podem ser primitivos ou derivados. Os tipos de dados primitivos são aqueles que não são definidos nos termos de outros tipos de dados, enquanto os tipos de dados derivados são aqueles que são definidos nos termos de outros tipos de dados. Para exemplificar, o XML *Schema* especifica que o tipo `float` é um conceito matemático que não pode ser definido nos termos de outros tipos de dados, enquanto o `integer` é um caso especial do decimal, o tipo de dado mais geral (BIRON & MALHOTRA, 2001).

Todo o tipo de dado primitivo é atômico. Isto é, o valor do tipo de dado não pode ser dividido. Por exemplo, o número 1 é um valor atômico. Já o tipo de dado derivado pode ser ou não atômico. Por exemplo, como visto anteriormente o `integer` é um tipo de dado derivado e com valor atômico. Por outro lado, um número telefônico é também um tipo de dado derivado, porém com o

valor não atômico; já que nesta forma é uma coleção de sete valores atômicos individuais (TIDWELL, 2001).

Tipos de dados são derivados principalmente por Restrição ou por Extensão (existem outras formas, mas essas são as mais comuns). Na derivação por restrição, o valor do tipo de dado é restrito de algumas formas. Por exemplo, um `integer` é uma derivação do tipo de dado decimal que permite uma escala de valores mais estreita de valores do que decimal. Um `integer`, em outras palavras, é reservado a conter um subconjunto restrito de valores decimais. A derivação por extensão significa que as várias restrições no tipo de dado base estão sendo elevadas para permitir valores adicionais que não seriam permitidos de outra maneira. Por exemplo, o tipo de dado de um número telefônico pode ser estendido para incluir o código de área (ibidem).

Os tipos de dados formam uma hierarquia que pode ser seguida a partir de um único tipo de dado primitivo e atômico chamado `anyType`. Todos os demais tipos de dados utilizados no XML *Schema* derivam desse único tipo primitivo. Existem dois gêneros de tipos de dados que podem ser derivados de `anyType`: tipos simples e tipos complexos. Os tipos simples representam todos os tipos de dados derivado e atômico embutidos no XML *Schema*. Isso inclui tipos como `string`, `integer` e `boolean`. Os tipos complexos representam todos os tipos de dados derivados e não atômicos (ibidem). No Anexo I, é possível observar a árvore hierárquica dos tipos de dados embutidos no XML *Schema*.

Biron & Malhotra (2001) afirma que o XML *Schema* suporta os seguintes tipos de dados primitivos: `string`, `boolean`, `decimal`, `float`, `double`, `duration`, `dateTime`, `time`, `date`, `gYearMonth`, `gYear`, `gMonthDay`, `gMonth`, `hexBinary`, `base64Binary`, `anyURI`, `Qname` e `Notation`.

## 3.4 UDDI

O UDDI (*Universal Description, Discovery and Integration*) é a definição de um conjunto de serviços apoiando a descrição e descoberta de: (i) negócios, organizações e outros provedores de *Web Services*; (ii) *Web Services* disponibilizados pelos provedores; e (iii) *interface* técnica que pode ser utilizada para acessar esses serviços. Baseada em um conjunto comum de padrões industriais, incluindo HTTP, XML, XML *Schema* e SOAP, a UDDI fornece a interoperabilidade e a infra-estrutura necessárias para *softwares* baseados em *Web Services* de ambos os ambientes: serviços

disponíveis publicamente e serviços disponíveis apenas internamente dentro da organização (BELLWOOD et al., 2002).

### 3.4.1 Função da UDDI

Segundo Ayala et al. (2002), a função da UDDI é fornecer os seguintes serviços básicos:

?? **Publicar:** Trata de como o provedor de *Web Service* registra a si mesmo e seu serviço no UDDI;

?? **Descobrir:** Trata de como o cliente pode localizar o *Web Service* desejado;

?? **Ligar:** Trata de como o cliente pode conectar e utilizar o *Web Service* localizado.

As associações que disponibilizam esses serviços e hospedam o registro global UDDI são chamadas de Operadores UDDI (*UDDI Operators*). Elas são responsáveis por sincronizar a informação de registro, sendo que a sincronização das informações de registro é chamada de replicação (*replication*). As empresas publicam seus serviços e as informações relacionadas utilizando esses operadores. Os dois operadores mais conhecidos são a Microsoft, que fornece o serviço de UDDI em `http://uddi.microsoft.com`, e a IBM, que fornece o mesmo tipo de serviço em `http://uddi.ibm.com` (AYALA et al., 2002).

No entanto, um identificador único *Universal Unique Identifier* (UUID) representa todo o nó operador. Para representar o ponto de replicação para cada um desses nós, é utilizado o atributo `SoapReplicationURL`, de forma que essa URL seja única em cada nó. O processo de replicação é essencial, permitindo assegurar que a informação armazenada em cada um dos operadores seja consistente (ibidem).

## 3.5 WSDL

### 3.5.1 Introdução

Segundo Ayala et al. (2002), WSDL (*Web Services Description Language*) é um formato XML para descrever a *interface* de um *Web Service* definindo um conjunto de operações suportadas

pelo servidor e o formato que os clientes necessitam utilizar quando requisitarem um serviço. Um arquivo WSDL atua como sendo um contrato entre o cliente e o serviço, pois efetiva a comunicação entre ambas as partes. O cliente requisitará o serviço enviando uma solicitação propriamente formatada em SOAP.

Cerami (2002) ressalta que o WSDL é uma plataforma independente de linguagem e, inicialmente, é utilizada (embora não exclusivamente) para descrever um serviço SOAP. Esse autor também enfatiza que, com o uso do WSDL, o cliente pode localizar um *Web Service* invocando qualquer uma das funções públicas disponíveis. Dessa forma, pode-se dizer que o WSDL representa uma base fundamental para a arquitetura de *Web Service*, pois disponibiliza uma linguagem comum para descrever os serviços e a plataforma de modo a haver a integração automática desses serviços.

Exemplificando o funcionamento do WSDL, pode-se imaginar a criação de um *Web Service* que ofereça as cotações na bolsa de valores para os compradores. Dessa forma, será necessária a criação de um arquivo WSDL que descreverá o serviço. Esse arquivo será alocado no provedor do *Web Service* ou publicado em um repositório UDDI. Os clientes interessados no serviço, primeiramente, terão que obter uma referência desse arquivo utilizando uma pesquisa ao repositório ou pegando-a diretamente do provedor. Após compreender a referência, deverá ser criado um *Web Service* solicitante que invocará o serviço fazendo uma requisição baseada em SOAP utilizando `HTTP post`. O servidor validará a requisição executando-a ou passando-a para o serviço adequado. O resultado, que é o preço das ações na bolsa de valores, é então retornado para o cliente como uma resposta no formato SOAP. Na Figura 2, pode-se observar o funcionamento do WSDL (AYALA et al., 2002).

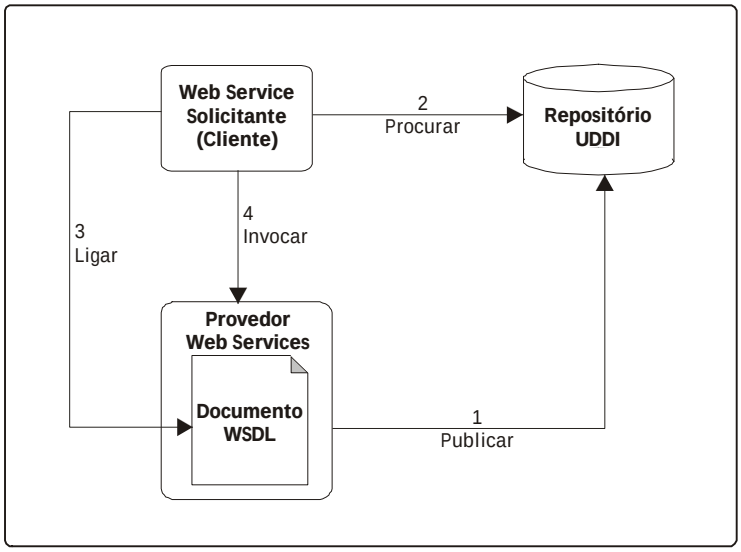



Figura 2: Funcionamento do WSDL.
Fonte: Ayala et al. (2002, p.40).

Conforme Cerami (2002), o WSDL apresenta quatro partes importantes de dados:

?? Informação da *interface* descrevendo todas as funcionalidades disponíveis publicamente;

?? Informações de tipos de dados para todas as mensagens de requisição ou mensagens de respostas;

?? Informação obrigatória sobre o protocolo de transporte a ser utilizado;

?? Informações de endereçamento para localizar um serviço especificado.

### 3.5.2 Estrutura de um documento WSDL

A estrutura de um documento WSDL é implementada utilizando XML *Schema* em conjunto com seus tipos suportados. A especificação para o documento WSDL fornece um conjunto com seis definições de elementos XML. O elemento raiz ou básico no documento WSDL é o elemento `definitions`. Os seis elementos aninhados são: `types`, `message`, `portType`, `binding`, `service` e `port`. (AYALA et al., 2002). No Anexo II, poderá ser consultado, na íntegra, o exemplo utilizado

durante esta seção. A Figura 3 apresenta de forma simples a estrutura de um documento WSDL contendo todos seus elementos, os quais são detalhados nas subseções a seguir.

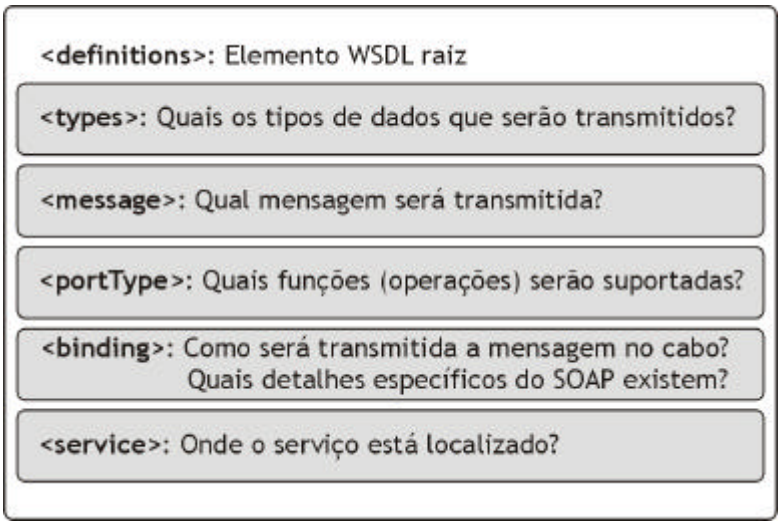



Figura 3: Estrutura de um arquivo WSDL.
Fonte: Cerami (2002).

### 3.5.2.1 Definitions

É obrigatório que o elemento `definitions` seja o elemento raiz de todo o documento WSDL. Ele define o nome do *Web Service* e declara múltiplos *namespaces* utilizados por todo o restante do documento (CERAMI, 2002).

O uso de *namespaces* é importante para diferenciar os elementos e permitir ao documento referenciar múltiplas especificações externas, incluindo também as especificações WSDL, SOAP e XML *Schema* (ibidem).

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
xmlns:tns="http://abcom.com/stocktrading.wsdl"
xmlns:xsd1="http://abcom.com/stocktrading.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://abcom.com/stocktrading.wsdl"
name="StockTrading">
```

Figura 4: Exemplo do elemento `definitions` de um arquivo WSDL.

Fonte: Ayala et al. (2002, p.79).

Como se pode observar, na Figura 4, a primeira linha define o arquivo como sendo um documento XML e especifica a codificação utilizada. A linha seguinte possui o elemento `definitions` que representa uma importante função na nomeação do serviço ou do documento WSDL. O nome do serviço é definido utilizando o atributo `name` (AYALA et al., 2002). O elemento `definitions` também especifica o atributo `targetNamespace`. Esse atributo é uma convenção XML *Schema* que permite ao documento WSDL referenciar a si próprio. No entanto, nota-se que, a especificação namespace não necessita que o documento realmente exista nesta localização; o importante é que deva ser especificado um valor único e diferente dos demais *namespaces* definidos. No final do elemento `definitions` existe um *namespace* padrão: `xmlns=http://schemas.xmlsoap.org/wsdl/`. Isso significa que todos os elementos sem prefixo *namespace*, como `message` ou `portType`, serão assumidos como parte do *namespace* WSDL padrão (CERAMI, 2002).

### 3.5.2.2 Types

Segundo Ayala et al. (2002), os elemento `types` descreve todos os tipos de dados utilizados entre o cliente e o servidor. O WSDL não é “amarrado” exclusivamente a um conjunto específico, mas ele utiliza a especificação XML *Schema* (apresentada na seção 3.2.2 XML *Schema*) como o conjunto de tipos padrão e trata isso como um conjunto de tipos nativo permitindo o máximo de interoperabilidade e neutralidade de plataforma. Se o serviço utilizar apenas tipos simples, como `string`, `integer` ou `float`, não é obrigatória a definição do elemento `types`.

O WSDL permite a conjuntos de tipos serem adicionados via elementos extensíveis. Um elemento extensível pode ser ajustado sob o elemento `types` para identificar o conjunto de

definição de tipos que foi utilizado e para fornecer um elemento *Container* XML para a definição de tipos. Como se pode observar na Figura 5, é especificado o conjunto de tipos no elemento `types` para os tipos de dados `string` e `float` (AYALA et al., 2002).

```
<types>
   <schema
targetNamespace="http://abcom.com/stocktrading.xsd"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
     <element name="StockRateRequest">
        <complexType>
           <all>
              <element name="StockScript"
type="string"/>
           </all>
        </complexType>
     </element>
     <element name="StockRate">
        <complexType>
           <all>
              <element name="rate" type="float"/>
           </all>
        </complexType>
     </element>
   </schema>
</types>
```

Figura 5: Exemplo de uso do elemento `type`.

Fonte: Ayala et al. (2002, p.83).

### 3.5.2.3 Message

O elemento `message` representa uma definição abstrata do dado antes de ser transmitido. Ele consiste de partes lógicas, cada qual ligada com uma definição dentro de algum conjunto `type`. A Figura 6 apresenta dois exemplos de definição do elemento `message` (ibidem).

```
<message name="GetStockRateInput">
   <part name="body" element="xsd1:StockRateRequest"/>
</message>
<message name="GetStockRateOutput">
   <part name="body" element="xsd1:StockRate"/>
</message>
```

Figura 6: Exemplos do elemento `message`.

Fonte: Ayala et al. (2002, p.85).

O atributo `element` refere-se a um elemento `xsd` utilizando um *namespace* qualificado, o prefixo `xsd` referencia o *namespace* para o XML *Schema*. O *namespace* qualificado refere-se ao elemento `type` no documento `xsd1:StockRateRequest`. O prefixo `xsd1` necessita ser definido no bloco de cabeçalho no início do documento WSDL. Um nome único é determinado pelo atributo `name` do elemento `message` entre todas as mensagens definidas dentro do documento WSDL. É importante ressaltar que o elemento `message` pode possuir mais que um elemento `part` (AYALA et al., 2002).

```
<message name="requisicaoOla">
   <part name="primeiroNome" element="xsd1:String"/>
</message>
```

Figura 7: Exemplo de um elemento `message` com atributo `type`.

Fonte: Cerami (2002).

O atributo `type` do elemento `part` especifica um tipo de dado de um XML *Schema*. O valor do atributo `type` precisa ser especificado como um tipo `Qname` do XML Schema e isso significa que o valor do atributo precisa ser um *namespace* qualificado. No exemplo apresentado na Figura 7, o atributo `type primeiroNome` é ajustado como `xsd:string`, que deve ser definido dentro do elemento `definitions` (CERAMI, 2002).

### 3.5.2.4 portType e Operation

Um `portType` é um conjunto de operações publicadas ou expostas por um *Web Service*, onde cada uma dessas operações refere-se a uma mensagem de entrada (*input*) ou saída (*output*). O elemento `operation` é o elemento filho do elemento `portType`. O elemento `operation` também pode possuir opcionalmente o atributo `parameterOrder` que define a ordem em que os `part names` serão listados quando formar a chamada RPC (AYALA et al., 2002).

Segundo Ayala et al. (2002), o elemento `operation` define a mensagem como sendo para envio e/ou recepção. Essas mensagens são definidas utilizando as expressões *input* e *output* no elemento `operation`. A Figura 8 apresenta uma operação com as duas expressões.

```
<portType name="StockTradingPortType">
   <operation name="GetStockRate">
      <input message="tns:GetStockRateInput"/>
      <output message="tns:GetStockRateInput"/>
   </operation>
</portType>
```

Figura 8: Exemplo de uso dos elementos `portType` e `operation`.

Fonte: Ayala et al. (2002, p.86).

O WSDL suporta quatro formas básicas de operação: (i) sentido único; (ii) requisição e resposta; (iii) solicitação; e (iv) resposta e notificação, descritas a seguir (CERAMI, 2002):

?? **Sentido único**: O cliente envia a mensagem para o serviço sem nenhuma resposta do serviço. A operação possui um único elemento `input`;

?? **Requisição e resposta**: O serviço recebe a mensagem e envia a resposta. A operação possui um elemento `input` seguido por um elemento `output`. Também é possível especificar o encapsulamento de erros;

?? **Solicitação e resposta**: O serviço envia a mensagem e recebe uma resposta. A operação possui um elemento `output` seguido por um elemento `input`. Também é possível especificar o encapsulamento de erros;

?? **Notificação**: O serviço envia a mensagem para o cliente. A operação possui um único elemento `output`.

A forma normalmente utilizada em serviços SOAP é a de requisição e resposta. Essas formas de operação são apresentadas na Figura 9.

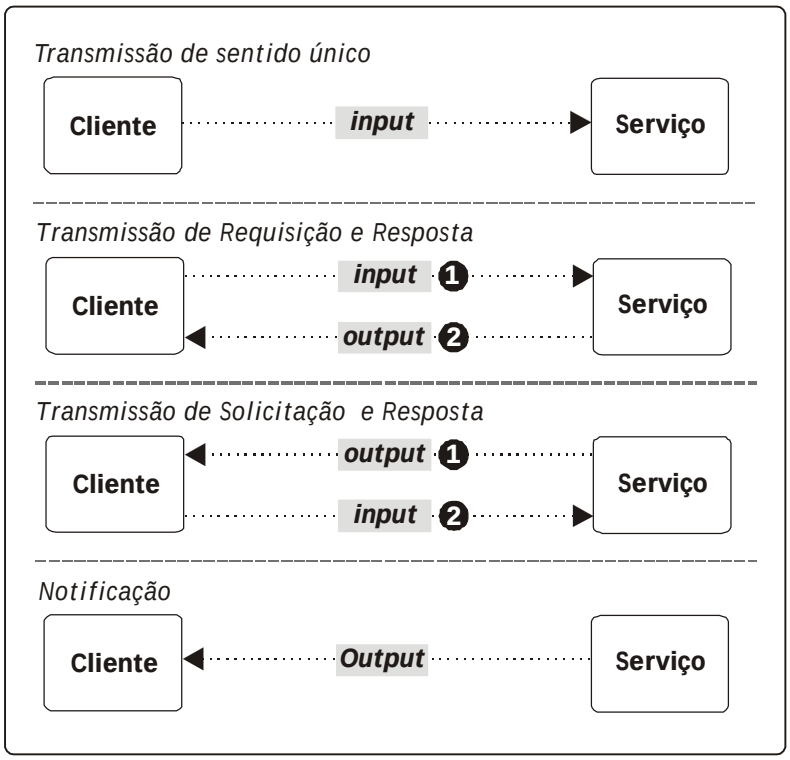



Figura 9: Formas de operação suportadas pelo WSDL.
Fonte: Cerami (2002).

### 3.5.2.5 Binding

Um elemento `binding` é definido como um canal que é utilizado pelo serviço para comunicar-se com o cliente. Esse elemento representa uma função importante na estrutura do *Web Service*, pois é independente de qualquer mecanismo de solicitação. Os mecanismos padrão utilizados nesse elemento são o HTTP, MIME (*Multi-purpose Internet Mail Extension*) e SOAP (AYALA et al., 2002).

Na Figura 10, pode-se observar que o nome do `binding` é `StockTradingSoapBinding`. O atributo `type` refere-se ao *namespace* qualificado do elemento `portType`. O *namespace* `tns` é definido anteriormente no documento. Neste exemplo, o SOAP é utilizado como mecanismo de acesso e, como pode-se observar, o elemento `binding` é seguido por um elemento `soap:binding` (ibidem).

```
<binding name="StockTradingSoapBinding"
type="tns:StockTradingPortType">
   <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="GetStockRate">
      <soap:operation
soapAction="http://abcom.com/GetStockRate"/>
      <input>
         <soap:body use="literal"/>
      </input>
      <output>
         <soap:body use="literal"/>
      </output>
   </operation>
</binding>
```

Figura 10: Exemplo de uso do elemento `binding`.

Fonte: Ayala et al. (2002, p.88).

O WSDL inclui suporte ao SOAP embutido. Isso permite que se possa especificar detalhes específicos do SOAP, incluindo cabeçalhos, codificações de estilo e o cabeçalho HTTP `soapAction`. Os elementos de extensão do SOAP incluem segundo Cerami (2002): soap:binding, soap:operation e soap:body.

3.5.2.5.1 soap:binding

Este elemento indica que o `binding` se tornará disponível por meio de SOAP. O atributo `style` indica o estilo completo do formato da mensagem SOAP. Os valores suportados pelo atributo `style` podem ser: `rpc` ou `document`. Se for atribuído o valor `rpc`, então todos os parâmetros enviados para o serviço irão ser "envelopados" em um elemento XML. Esse elemento receberá o mesmo nome que o elemento `operation`. O valor padrão para esse atributo é `document`. Nesse caso, a mensagem será diretamente "envelopada" sem nenhuma modificação.

O atributo `transport` indica o transporte da mensagem SOAP. O valor `http://schemas.xmlsoap.org/soap/http` indica que o transporte será feito por SOAP HTTP, enquanto que `http://schemas.xmlsoap.org/soap/smtp` indica que o transporte será feito por SOAP SMTP.

#### 3.5.2.5.2 soap:operation

Este elemento possui o atributo `soapAction` que envia uma solicitação ao serviço. Esse atributo contém um campo de cabeçalho que armazena informações sobre o serviço requisitado para invocação. Esse é um campo obrigatório para a requisição SOAP enviado via HTTP. O trabalho do solicitante torna-se fácil porque ele faz uso desse campo para invocar o serviço requisitado. Esse campo também pode atuar como um identificador único que o servidor utiliza para direcionar a requisição ao serviço correto.

#### 3.5.2.5.3 soap:body

Este elemento permite que seja especificado os detalhes das mensagens de input e output. Ele possui um atributo `encodingStyle` que especifica qual formato atual é utilizado para a codificação. Existem dois esquemas básicos SOAP: o `encoded`, que é um estilo de codificação de dados conforme a especificação SOAP; e o `literal`, que especifica que o dado é simplesmente enviado e recebido como um simples documento XML sem nenhuma codificação.

### 3.5.2.6 Service e Port

O elemento `service` define o endereço para invocar um serviço específico, normalmente incluindo um URL (*Uniform Resource Locator*) (CERAMI, 2002). O elemento `service` possui um elemento filho chamado `port`, que por sua vez possui toda a informação requisitada pelo solicitante. O elemento `port` consiste de um tipo específico de ligação para aquele serviço. O atributo `binding` refere-se ao elemento `binding` apresentado anteriormente. O elemento `soap:address` indica para o endereço atual do serviço. Pode-se observar um exemplo na Figura 11, juntamente com o elemento `service` e a localização do serviço (AYALA et al., 2002).

```
<service name="StockTradingService">
   <documentation>FIRST SERVICE</documentation>
   <port name="StockTradingPort"
binding="tns:StockTradingSoapBinding">
      <soap:address location="http://abcom.com/stocktrading"/>
   </port>
</service>
```

Figura 11: Exemplo de uso dos elementos `service` e `port`.

Fonte: Ayala et al. (2002, p.93).

Conforme a especificação WSDL é possível haver mais de um elemento `port` tendo diferentes tipos de ligações; desta forma, é possível efetuar ligações para SOAP e HTTP no mesmo elemento `port`. O propósito desta implementação é permitir a um *Web Service* acessar através de múltiplas portas ao mesmo tempo (AYALA et al., 2002).

## 3.6 Considerações

Como apresentado nessa seção, os *Web Services* sáo definidos por um documento WSDL, que é uma estrutura em XML para descrever os métodos e atributos utilizados. Para implementar um *Web Service* é possivel utilizar dois protocolos diferentes: SOAP e XML-RPC. Estes procotolos serão descritos no capítulo a seguir, em maiores detalhes.

# 4 PROTOCOLOS DE WEB SERVICES

## 4.1 Introdução

Existem dois protocolos principais de *Web Services*: SOAP e XML-RPC. Embora ambos possam executar Chamada Remota de Processos (RPC - *Remote Procedure Call*), estes possuem formas diferentes de tratar suas conexões. O SOAP permite o gerenciamento do estado das conexões (*stateful*), dessa forma cada chamada RPC é gerenciada armazenando informações sobre a conexão e as requisições ao serviço. O protocolo XML-RPC não permite o gerenciamento do estado das conexões (*stateless*), isso significa que cada requisição é gerenciada de forma independente, não armazenando informações entre as requisições (AYALA et al., 2002).

## 4.2 XML-RPC

O XML-RPC é uma especificação e um conjunto de implementações que permitem aos programas “rodarem” em sistemas operacionais e ambientes diferentes para fazerem chamadas de procedimentos pela Internet. Essa chamada de procedimento remoto usa HTTP como transporte e XML como codificação. O XML-RPC é projetado para ser tão simples quanto o possível, enquanto permite estruturas de dados complexas serem transmitidas, processadas e retornadas (USERLAND, 2001).

O padrão XML-RPC permite integração com linguagens de programação distintas e, também, permite interoperabilidade entre diferentes sistemas operacionais fazendo com que várias aplicações integrem-se perfeitamente dentro de uma única aplicação distribuída. Os programas escritos em XML-RPC podem ser desenvolvidos em qualquer linguagem de programação incluindo a linguagem C. A tradução do código XML para qualquer uma dessas linguagens é transparente para o desenvolvedor. A troca de informação é feita por meio de pacotes de solicitação e resposta codificado em XML (AYALA et al., 2002).

O XML-RPC baseia-se em uma chamada *web* do tipo `HTTP Post` contendo uma requisição, e a resposta correspondente contendo o resultado. A requisição possui o nome de um método e a lista de valores dos parâmetros. A resposta deve conter exatamente um valor de retorno, ou uma condição de erro. A limitação a um simples valor não é um problema, pois tipos compostos podem ser utilizados para expressar múltiplos valores (DUMBILL, 1999). É importante ressaltar que o XML-RPC não fornece nenhum meio de segurança por si só (LATTEIER, 2000).

### 4.2.1 Arquitetura do XML-RPC

Em um sistema típico cliente-servidor, o cliente efetua uma chamada de processamento remoto para um programa que esteja "rodando" em uma máquina remota. No XML-RPC, solicitações RPC são codificadas em documentos XML que são enviados a um servidor por meio de HTTP. Quando o servidor recebe, ele decodifica o documento, executa o procedimento solicitado, "empacota" o resultado (se existir) em um documento XML e o retorna ao cliente também por meio de HTTP. O cliente decodifica a resposta e continua sua execução (AYALA et al., 2002).

### 4.2.2 Tipos de Dados

Como se pode observar, na Tabela 1, o número de tipos de dados suportados é realmente pequeno. O XML-RPC não permite o uso de tipos de dados definidos pelo usuário (ibidem).

Tabela 1: Tipos de dados suportados pelo XML-RPC.

| **Tipo** | **Descrição** |
|---|---|
| `Int` | Valor inteiro de 32 bits (4 *bytes*). |
| `string` | Uma string ASCII, que pode conter *bytes* nulos (atualmente, várias implementações XML-RPC suportam *unicode*). |

| Tipo | Descrição |
|---|---|
| `boolean` | Verdadeiro (1) ou Falso (0) |
| `double` | Número de ponto flutuante com precisão dupla. |
| `dateTime.iso8601` | Data e Hora. Infelizmente, desde que o XML-RPC proibiu o uso de *timezones*, este tipo é inútil. |
| `base64` | Dados binários brutos (*raw*) de qualquer tamanho. |
| `array` | Um vetor de valores de uma dimensão. |
| `struct` | Uma coleção de pares de valores e chaves. As chaves são *strings*; os valores podem ser de qualquer tipo. |

Fonte: Kidd (2001).

### 4.2.3 Formato da Requisição

Um exemplo de um documento contendo uma solicitação XML-RPC é apresentado na Figura 12 (WINER, 1999).

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
<methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value><i4>41</i4></value>
      </param>
   </params>
</methodCall>
```

Figura 12: Exemplo de uma solicitação XML-RPC.

Fonte: Winer (1999).

Neste exemplo, o primeiro bloco de código refere-se ao cabeçalho e o segundo refere-se aos dados transportados (chamada de procedimento e valores de parâmetros), descritos a seguir:

?? O formato da URI na primeira linha do cabeçalho pode ser variável. Por exemplo, ela pode ser vazia ou uma simples barra ( / ) se o servidor estiver apenas tratando de chamadas XML-RPC. Porém, se o servidor tratar várias requisições de HTTP, é permitido à URI

ajudar no roteamento de requisições ao código que tratará as requisições XML-RPC. No exemplo, a URI é /RPC2;

?? Um `User-Agent` e um *Host* precisam ser especificados. Um `User-Agent` é um termo técnico para navegador Internet (*Browser*); e um *Host* é o nome do computador na rede;

?? O `Content-Type` deve ser definido como `text/xml`, indicando que o tipo de conteúdo do arquivo é um formato XML;

?? O `Content-length` é o tamanho da solicitação XML-RPC; deve ser especificado e possuir o valor exato de caracteres.

Como pode ser observado no cabeçalho, o conteúdo do documento está no formato XML. Ele possui apenas uma estrutura chamada `methodCall`. Essa estrutura possui um sub-item `methodName`, que é uma *string* contendo o nome do método a ser chamado. Se a chamada de procedimento possuir parâmetros, e neste exemplo possui, o `methodCall` deve possuir um sub-item `params`. Esse sub-item pode conter quantos `param` forem necessários, e cada `param` deve possuir um `value`. Este exemplo possui apenas um valor.

### 4.2.4 Formato da Resposta

A Figura 13 apresenta a resposta da solicitação efetuada no exemplo anterior (WINER, 1999).

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
      </param>
   </params>
</methodResponse>
```

Figura 13: Exemplo de uma resposta XML-RPC.

Fonte: Winer (1999).

A resposta também é dividida em dois blocos: o cabeçalho e o corpo da mensagem com a resposta em XML. Na primeira linha, a menos que exista um erro de baixo nível, o retorno sempre será "`200 OK`".

O corpo da mensagem é a resposta estruturada em XML. O elemento `methodResponse` possui um único elemento do tipo `params`. Este, por sua vez, é composto pelo elemento `param`, que por fim possui a resposta dentro do elemento `value`.

Em caso de erro, o `methodResponse` pode conter o elemento `fault`, que, dentro de sua estrutura, possuirá outros dois elementos: `faultCode`, com um valor `integer` indicando o tipo de erro, e `faultString`, que é uma `string` com a descrição do erro. O `methodResponse` não poderá conter ambos: `fault` e `params`. O exemplo de uma resposta com falha é apresentado na Figura 14.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
            <value><int>4</int></value>
        </member>
        <member>
          <name>faultString</name>
            <value><string>Too many parameters.</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

Figura 14: Exemplo de uma resposta XML-RPC com falha.

Fonte: Winer (1999).

## 4.3 SOAP

O SOAP (*Simple Object Access Protocol*) é um protocolo simples para troca de informações em um ambiente distribuído. Ele é baseado em XML e consiste de três partes: um envelope que

define uma estrutura para descrever o que está dentro da mensagem e como processá-la; um conjunto de regras codificadas para expressar ocorrências de tipos de dados definidos pela aplicação; e a convenção para representar as chamadas remotas de processos e as respostas. O SOAP pode ser utilizado potencialmente em conjunto com uma variedade de outros protocolos (BOX et al., 2000).

O SOAP é um protocolo de mensagens baseado em XML utilizado para chamadas remotas de procedimentos (RPC). A especificação SOAP descreve a forma de representação de chamadas RPC dentro de mensagens SOAP, além de um formato de mensagem para comunicação máquina-a-máquina. O SOAP resolve problemas encontrados anteriormente no RPC. Ele também define elementos para especificar o nome do método, seus parâmetros e os tipos de retorno. A vantagem do SOAP é que as mensagens são embutidas em requisições HTTP, permitindo que elas penetrem facilmente pelos *firewalls*. Por ser um padrão aberto, o SOAP permite interoperabilidade entre sistemas RPC heterogêneos que implementem o suporte a SOAP (AYALA et al., 2002).

O SOAP pode ser utilizado em combinação com uma variedade de protocolos Internet existentes incluindo HTTP, SMTP, MIME e pode suportar uma larga cadeia de aplicações de sistemas de mensagem para RPC (BOX et al., 2000).

### 4.3.1 Arquitetura

Segundo Box et al. (2000), a implementação do SOAP pode ser otimizada para explorar a única característica de um sistema de rede particular. Por exemplo, a ligação HTTP disponibiliza para mensagens de resposta SOAP de serem entregues como respostas HTTP, utilizando a mesma conexão que a requisição destinada. Apesar do protocolo ao qual o SOAP é limitado, as mensagens são roteadas adiante para um assim chamado "caminho da mensagem", que permite levar em conta o processamento em um ou mais intermediários em acréscimo ao destino final.

A aplicação SOAP recebendo a mensagem SOAP deve processar a mensagem executando as seguintes ações na ordem em que estão listadas (ibidem):

1. Identificar todas as partes da mensagem SOAP destinadas à aplicação;

2. Verificar se todas as partes identificadas como obrigatórias no passo 1 são suportadas pela aplicação para essa mensagem e processá-las conseqüentemente. Se esse não é o caso então descartar a mensagem. O processador talvez ignore partes opcionais identificadas no passo 1 sem afetar o resultado do processamento;
3. Se a aplicação SOAP não for o destino final da mensagem, então ela remove todas as partes identificadas no passo 1 antes de repassar a mensagem.

Para processar uma mensagem ou parte dela, é preciso que o processador SOAP entenda, entre outras coisas, dos padrões de troca em uso (caminho único, requisição/resposta, *multicast*, entre outros), a regra do receptor no padrão, a atividade (se possuir), o mecanismo RPC, a representação ou codificação dos dados, da mesma forma que outras semânticas necessárias para o correto processamento (ibidem).

### 4.3.2 Mensagem SOAP

Uma mensagem SOAP é um documento XML que consiste obrigatoriamente de um envelope e de um corpo SOAP e, opcionalmente, de um cabeçalho SOAP. Esse documento XML é conhecido como mensagem SOAP (BOX et al., 2000). Como apresentado na Figura 15, a mensagem SOAP possui algumas regras de construção.

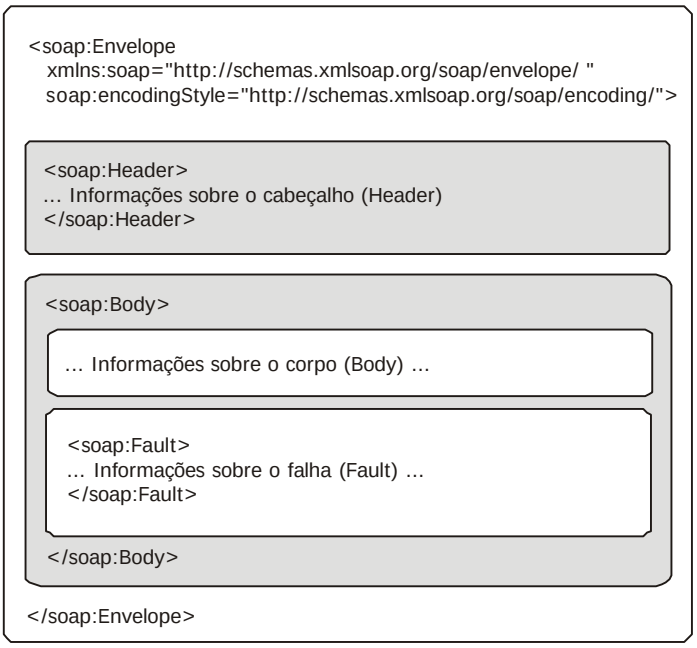



Figura 15: Composição de uma mensagem SOAP.
Fonte: Cunha (2002).

#### 4.3.2.1 Envelope

Um envelope (`envelope`) é o elemento raiz do documento XML representando a mensagem. O elemento envelope pode conter declarações de *namespace* e também atributos adicionais. O *namespace* referente ao SOAP 1.1 deve ser `http://schemas.xmlsoap.org/soap/envelope`. Esse *namespace* indica a versão do SOAP que está sendo utilizada. Para a versão em desenvolvimento do SOAP 1.2, o identificador *namespace* é `http://www.w3.org/2001/06/soap-envelope`. O *namespace* padrão para codificação e tipos de dados (`encodingStyle`) é `http://schemas.xmlsoap.org/soap/encoding/`. Essa URI especifica as regras de serialização definidas pela especificação SOAP (BOX et al., 2000). A serialização é uma técnica para representação de modelos de dados não sintáticos baseados nos tipos de dados do XML *Schema*.

#### 4.3.2.2 Cabeçalho (Header)

Após o elemento envelope, o documento pode conter, opcionalmente, o elemento cabeçalho (*header*). O cabeçalho é um mecanismo genérico que adiciona características à mensagem SOAP de uma maneira descentralizada sem um prévio entendimento entre as partes da comunicação. Entre os exemplos típicos de atributos que podem ser implementados como entradas no cabeçalho são: autenticação, gerenciamento de transações, entre outros. Se definido, o cabeçalho deve ser o primeiro elemento filho direto do envelope SOAP (BOX et al., 2000).

Uma entrada no cabeçalho é identificada pelo seu nome de elemento totalmente qualificado, que consiste do *namespace* URI e o nome do local. Todos os elementos imediatamente filhos do elemento cabeçalho devem ser um *namespace* qualificado (ibidem). A Figura 16 ilustra o uso de atributos no elemento cabeçalho.

```
<soap:Header>
   <m:UserInfo xmlns:m="http://www.wrox.com/Info/">
      <m:UserID>123456</m:UserID>
      <m:Password>abcdef</m:Password>
   </m:UserInfo>
</soap:Header>
```

Figura 16: Exemplo do elemento cabeçalho com uso de atributos.

Fonte: Ayala et al. (2002, p.60).

O elemento cabeçalho pode possuir mais dois atributos definidos: `Actor` e `mustUnderstand`.

4.3.2.2.1 Atributo Actor

Uma mensagem SOAP pode viajar da origem até seu destino final, potencialmente passando por um conjunto de intermediários SOAP durante o caminho da mensagem. Um intermediário SOAP é uma aplicação que tem a capacidade de receber e de repassar uma mensagem SOAP. Tanto o destinatário final quanto o intermediário devem ser identificados por um URI (BOX et al., 2000).

O atributo `actor` define o destino final dos elementos do cabeçalho. O destino final aqui é o intermediário que receberá a mensagem SOAP para algum processamento. O atributo `actor` especifica a URI do intermediário no qual a informação do cabeçalho SOAP é o alvo. Uma vez que o processamento for concluído esta informação é removida. Se a URI for `http://schemas.xmlsoap.org/soap/actor/next` isto indica que o elemento cabeçalho é destinado à primeira aplicação SOAP que processar a mensagem. Caso seja omitido o atributo `actor`, o receptor será o destino final da mensagem SOAP (AYALA et al., 2002).

4.3.2.2.2 Atributo mustUnderstand

O atributo `mustUnderstand` é utilizado para indicar se determinada entrada do cabeçalho deve ser processada obrigatoriamente ou opcionalmente no destino final. O valor do atributo `mustUnderstand` pode ser verdadeiro (1) ou falso (0). Caso não seja definido esse atributo, o valor padrão é falso (0) (BOX et al., 2000).

O exemplo da Figura 17 apresenta o uso dos dois atributos: `actor` e `mustUnderstand`. Neste exemplo, as informações contidas no elemento `m:UserInfo` serão destinadas para o atributo `actor` (`http://www.wrox.com/auth`) que é o intermediário SOAP que utilizará as informações contidas no cabeçalho para validar o usuário. O atributo `mustUnderstand` dos elementos `m:UserID` e `m:Password` estão ajustados para 1 (verdadeiro) o que significa que o destino final desses elementos (a URI contida dentro do atributo `actor`) devem compreender o processo de validação não podendo simplesmente ignorá-lo(AYALA et al., 2002).

```
<soap:Header>
   <m:UserInfo xmlns:m="http://www.wrox.com/Info/"
      soap:actor="http:/www.wrox.com/auth/">
      <m:UserID
soap:mustUnderstand="1">123456</m:UserID>
      <m:Password
soap:mustUnderstand="1">abcdef</m:Password>
      <m:email
soap:mustUnderstand="0">neto@univali.br</m:email>
   </m:UserInfo>
</soap:Header>
```

Figura 17: Exemplo de uso dos atributos `actor` e `mustUnderstand`.

Fonte: Ayala et al. (2002, p.63).

#### 4.3.2.3 Corpo (Body)

O elemento corpo (`body`) provê um simples mecanismo para a troca de informações obrigatórias destinado ao receptor da mensagem. Usos típicos do elemento corpo incluem organização de chamadas RPC e informações de erro. O elemento corpo deve ser o primeiro elemento após o elemento cabeçalho. Caso não tenha sido definido um elemento cabeçalho, então o elemento corpo torna-se o primeiro elemento da mensagem, após o elemento envelope. O corpo, assim como o envelope, é um elemento obrigatório (BOX et al., 2000).

```
<soap:body>
   <GetStockPrice>
      <Symbol>IBM</Symbol>
   </GetStockPrice>
</soap:Body>
```

Figura 18: Exemplo do elemento corpo.

Fonte: Ayala et al. (2002, p.61).

Todos os elementos filhos do elemento corpo são chamados de entradas (`body entries`). Cada entrada é codificada como um elemento independente dentro do elemento corpo. No exemplo apresentado na Figura 18, as entradas `GetStockPrice` e `Symbol` são específicas da aplicação e não fazem parte do padrão SOAP. Nesse caso, a entrada `GetStockPrice` representa o nome do método a ser chamado e a entrada `Symbol` representa um simples parâmetro para o método (AYALA et al., 2002). O SOAP define um elemento para o corpo que é o elemento `fault`, utilizado para relatar erros (BOX et al., 2000).

#### 4.3.2.4 Falha (Fault)

O elemento SOAP Falha (`Fault`) é utilizado para transmitir erros e/ou informações dentro da mensagem SOAP. Se presente, o elemento Falha deve aparecer como uma entrada no corpo da mensagem e não deve aparecer mais que uma vez dentro do elemento corpo (AYALA et al., 2002). O elemento Falha define os seguintes sub-elementos: `faultcode`, `faultstring`, `faultactor` e `detail`, descritos a seguir (BOX et al., 2000):

?? **faultcode**: Este elemento contém a informação que indica o tipo de erro. O conteúdo deste elemento deve ser um *namespace* qualificado. Essa informação é intencionada para a aplicação e, portanto, não pode ser apresentada diretamente ao usuário final;

?? **faultstring**: Este é outro elemento filho do elemento falha que é destinado a fornecer uma informação de forma "legível" relacionada à falha. Essa informação é intencionada ao usuário final e não possui nenhuma utilidade para o processamento da aplicação. O elemento `faultstring` é obrigatório no elemento Falha;

?? **faultactor**: Este elemento apresenta informações de quem causou a falha dentro do caminho da mensagem. É similar ao atributo `actor`, mas ao invés de indicar o destino das entradas do cabeçalho, ele indica a origem da falha;

?? **detail**: Apesar de ser possível dispor de informações completas relativas à falha utilizando os elementos anteriores, é possível incluir informações adicionais ao elemento `detail`. Essas informações podem ser usadas para depurar a aplicação. O servidor deve incluir esse elemento na mensagem de resposta se o erro for gerado durante o processamento do corpo da mensagem SOAP.

### 4.3.3 Tipos de Dados

O estilo de codificação é baseado em sistemas de tipos simples, que é generalizado de características comuns encontradas em tipos de sistemas de linguagens de programação, banco de dados e dados semi-estruturados. Um tipo pode ser simples (escalar) ou composto, construído como um composto de várias partes, cada uma com um tipo (BOX et al., 2000).

#### 4.3.3.1 Tipos Simples

Para tipos simples o SOAP adota todos os tipos encontrados na definição do XML *Schema* (apresentado na seção 3.2.2 XML *Schema*). Os tipos simples podem ser representados em um simples elemento na mensagem SOAP, conforme a Figura 19 (AYALA et al., 2002).

```
<EmpName xsi:type="xsd:string">Mike</EmpName>
```

Figura 19: Exemplo de representação de tipos simples em um único elemento.
Fonte: Ayala et al. (2002, p.67).

Na Figura 20, é apresentado outro exemplo de tipos simples, porém com o uso de *schema*. Neste caso, são definidos previamente os elementos com seus tipos correspondentes, que são utilizados na continuação do documento (ibidem).

```
<xs:schema xmlns:xs="http://w3.org/1999/XMLSchema" >
  <xs:element name="EmpCode" type="xs:int" />
  <xs:element name="EmpName" type="xs:string" />
  <xs:element name="Salary" type="xs:float" />
</xs:schema>

<EmpCode>4354</EmpCode>
<EmpName>Mike</EmpName>
<EmpSalary>2500</EmpSalary>
```

Figura 20: Exemplo de representação de tipos simples em *schema*.
Fonte: Ayala et al. (2002, p.67).

##### 4.3.3.1.1 Strings

O SOAP copia o tipo de dados `string` definido no XML *Schema*. Este tipo possivelmente não é o mesmo encontrado em qualquer banco de dados ou linguagem de programação, portanto ,talvez não suporte alguns dos caracteres(ibidem).

##### 4.3.3.1.2 Enumerations

O modelo SOAP adota o mecanismo `enumeration` suportado pelo XML *Schema*. O `enumeration` é um conjunto de valores distintos com o mesmo tipo base. Por exemplo, um

conjunto de moedas ("Dollar", "Real" e "Euro") pode ser definido como um `enumeration`. A Figura 21 apresenta uma definição do tipo `enumeration` (AYALA et al., 2002).

```
<xs:simpletype name="Moedas" >
  <xs:restriction base="xs:string" >
    <xs:enumeration value="Dollar" />
    <xs:enumeration value="Real" />
    <xs:enumeration value="Euro" />
 </xs:restriction>
<xs:simpleType>
```

Figura 21: Exemplo de uso do tipo simples `enumeration`.

Fonte: Ayala et al. (2002, p.68).

#### 4.3.3.1.3 Array de Bytes

Tanto o SOAP quanto o XML *Schema* disponibilizam o tipo para representação de *array* de *bytes*. O XML *Schema* utiliza a codificação `base64` que é uma representação recomendada para dados binários. Utilizando esse tipo, é possível representar o dado binário na forma de um *array* de *bytes* na mensagem SOAP.

### 4.3.3.2 Tipos Complexos

Segundo Ayala et al. (2002), é necessário algum mecanismo para representar as entidades complexas do mundo real na forma de mensagens XML. O conceito é parecido ao encontrado em linguagens de programação onde são utilizadas estruturas e *arrays* para representar um dado logicamente próximo. Seguindo esse caminho, a codificação SOAP provê dois tipos compostos: Estruturas e *Arrays*.

#### 4.3.3.2.1 Estruturas

A estrutura é um valor composto que usa um assessor para representar os membros. O nome do assessor é a única distinção nos valores dos membros. Na mesma estrutura, dois assessores não podem possuir o mesmo nome. Na Figura 22 é apresentado um exemplo de registro de aluno, onde são armazenados o nome do aluno, o número de matrícula e o curso (ibidem).

```
<xs:element name="estudante"
            xmlns:xs='http://www.w3.org/2001/XMLSchema' >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Matricula" type="xs:int" />
      <xs:element name="nomeAluno" type="xs:string" />
      <xs:element name="Curso" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
<xs:element>
```

Figura 22: Exemplo de um XML *Schema* para a estrutura estudante.

Fonte: Ayala et al. (2002, p.70).

A instância da estrutura declarada no exemplo anterior pode ser observada no documento XML apresentado na Figura 23.

```
<xs:element name="estudante"
            xmlns:xs='http://www.w3.org/2001/XMLSchema' >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Matricula" type="xs:int" />
      <xs:element name="nomeAluno" type="xs:string" />
      <xs:element name="Curso" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
<xs:element>
```

Figura 23: Exemplo de uso da estrutura estudante.

Fonte: Ayala et al. (2002, p.70).

4.3.3.2.2 Arrays

Um *Array* é semelhante a uma estrutura. A diferença entre esses dois tipos está na forma de referenciar os membros. Na estrutura os membros são identificados pelo nome do seu assessor, porém, no caso dos *arrays*, o membro é identificado pela posição ordinal. No SOAP os *arrays* são definidos usando um tipo de dado especial: `enc:Array`. Pode-se observar um exemplo de uso do *array* na Figura 24. Neste exemplo, o *array* `myFavoriteNumbers` contém vários membros onde cada um é um valor do tipo `xsd:int`. Nota-se que o tipo `enc:Array` permite nomes de elementos não qualificados sem restrições (AYALA et al., 2002).

```
<element name="myFavoriteNumbers"
        type="enc:Array"/>

<myFavoriteNumbers
  enc:arrayType="xsd:int[2]">
   <number>3</number>
   <number>4</number>
</myFavoriteNumbers>
```

Figura 24: Exemplo de uso do tipo *Array*.

Fonte: BOX et al. (2000).

Conforme Ayala et al. (2002), todo elemento no *array* pode ser de tipo simples ou composto. Dessa forma, *arrays* podem ser utilizados para representar mais estruturas complexas na mensagem SOAP.

### 4.3.4 Encapsulando mensagens SOAP em HTTP

Utilizar o SOAP em conjunto com HTTP disponibiliza a vantagem de ser capaz de utilizar o formalismo e a flexibilidade descentralizada do SOAP com as valiosas características do conjunto HTTP. Transmitindo o SOAP no HTTP não significa que o SOAP cancelará a semântica existente do HTTP, mas que, de preferência, as semânticas do SOAP descrevam naturalmente a semântica do HTTP (BOX et al., 2000).

O SOAP naturalmente segue o modelo de mensagens de requisição/resposta do HTTP, fornecendo parâmetros de requisições SOAP em requisições HTTP e parâmetros de respostas SOAP em respostas HTTP. É importante notar que, no entanto os intermediários SOAP não são os mesmos que os intermediários HTTP. Aplicações HTTP devem usar o tipo de meio "`text/xml`" quando incluírem entidades SOAP no corpo das mensagens HTTP (ibidem).

#### 4.3.4.1 Requisição HTTP SOAP

Embora o SOAP talvez seja utilizado em combinação com uma variedade de métodos de requisição HTTP, essa ligação apenas define-o dentro de requisições `HTTP Post`. O campo `SOAPAction` no cabeçalho HTTP pode ser utilizado para indicar a intenção da requisição HTTP SOAP. O valor é um URI identificando o objetivo. O SOAP não impõem restrições no formato ou especificação da URI ou se ela pode ser resolvida. Um cliente HTTP deve usar este campo de

cabeçalho quando cria uma requisição HTTP SOAP (BOX et al., 2000). A Figura 25 apresenta uma requisição HTTP SOAP.

```
POST /StockQuote HTTP/1.1
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://electrocommerce.org/abc#MyMessage"
...
 Mensagem SOAP
...
```

Figura 25: Exemplo de um HTTP SOAP utilizando POST.

Fonte: Ayala et al. (2002, p.37).

### 4.3.4.2 Resposta HTTP SOAP

O HTTP SOAP segue a semântica de códigos de estado do HTTP para comunicar informações de estado no HTTP. Por exemplo, o código de estado 2xx indica que a requisição do cliente, incluindo o componente SOAP, foi recebida, entendida e aceita com sucesso. No caso de erros SOAP durante o processamento da requisição, o servidor HTTP SOAP deve criar uma resposta HTTP 500 (erro interno do servidor) e incluir a mensagem SOAP na resposta contendo o elemento SOAP falha (`fault`) indicando o erro de processamento do SOAP (BOX et al., 2000). Um exemplo de resposta HTTP contendo erro pode ser observada na Figura 26.

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
       <SOAP-ENV:Fault>
           <faultcode>SOAP-ENV:MustUnderstand</faultcode>
           <faultstring>SOAP Must Understand  Error</faultstring>
       </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 26: Exemplo de resposta HTTP com erro.

Fonte: Box et al. (2000).

## 4.4 Comparativo entre XML-RPC e SOAP

O XML-RPC é muito similar ao SOAP. Porém os clientes e servidores escritos para XML-RPC não são interoperáveis diretamente com os clientes e servidores SOAP. Mesmo que as mensagens XML possuam estruturas semelhantes, a diferença no formato da mensagem previne essa interoperabilidade (AYALA et al., 2002).

O SOAP e o XML-RPC são protocolos relacionados, onde ambos tentam homogeneizar a passagem de parâmetros para um *Web Service*. O SOAP é atualmente uma recomendação W3C (*World Wide Web Consortium*) e ativamente está sendo ampliada e desenvolvida por empresas como Microsoft e IBM. O Microsoft .NET é largamente induzido por ser capaz de acessar *Web Services* de forma transparente através de mensagens SOAP (APPLE, 2002).

O SOAP é um padrão emergente para chamada de procedimentos em objetos remotos. A diferença básica entre SOAP e XML-RPC é que os procedimentos do SOAP recebem parâmetros nomeados onde a requisição deve codificar os nomes de parâmetros do método dentro do XML. A ordem desses parâmetros é irrelevante, mas no XML-RPC a ordem é relevante e os parâmetros não são nomeados (AYALA et al., 2002).

O XML-RPC é uma especificação final que é menos detalhada e mais fácil de implementar que o SOAP. Ambos SOAP e XML-RPC trabalham trocando um conjunto de valores (`strings`, `dates`, `arrays` e `binary`) dentro do XML para a transmissão. O XML-RPC é definido para operar em conexões sobre HTTP, enquanto o SOAP descreve um formato de envelope para uma requisição RPC que pode ser enviada através de HTTP, SMTP ou algum outro protocolo (APPLE, 2002).

Tanto o SOAP quanto o XML-RPC suportam a passagem de dados binários no documento XML com o uso da codificação `base64` (ibidem).

Conforme Ayala et al. (2002), quando o XML-RPC foi criado, o XML *Schema* ainda não estava pronto para uso. Dessa forma, foi implementado um conjunto simples e fixo de tipos de dados sendo de fácil implementação e verificação, ao contrário do SOAP que utiliza o XML *Schema* para definir seus tipos. Assim, o XML-RPC não suporta referências na definição de seus tipos. Por outro lado, as mensagens resultantes são mais detalhadas que as equivalentes em SOAP.

A Tabela 2 apresenta um comparativo entre algumas características do SOAP e do XML-RPC.

Tabela 2: Comparativo entre algumas características do XML-RPC e do SOAP.

| **Características** | **SOAP** | **XML-RPC** |
|---|---|---|
| *Arrays* e estruturas nomeadas | Sim | Não |
| Manusear falhas detalhadas | Sim | Sim |
| Curva de aprendizado curta | Não | Sim |
| Conjunto de caracteres especificados pelo desenvolvedor (UTF-8, US-ASCII, UTF-16) | Sim | Não |
| Tipos de dados definidos pelo desenvolvedor | Sim | Não |
| Pode ser especificado o receptor | Sim | Não |
| Necessita o entendimento do Cliente | Sim | Não |
| Estruturas | Sim | Sim |
| *Arrays* | Sim | Sim |
| Passagem de arquivos binários (`base64`) | Sim | Sim |
| Pode ser descrito por um WSDL | Sim | Não |

Fonte: Rhodes (2003).

## 4.5 Segurança

Segundo Ayala et al. (2002), existem muitos protocolos e sistemas de segurança que podem ser implementados para que se possa obter uma arquitetura de sistema seguro. Faz-se necessário a escolha de métodos adequados, para garantir a segurança na transação efetuada entre dois pontos. Durante o planejamento ou desenvolvimento de um *Web Service*, é importante considerar as técnicas descritas a seguir para garantir maior segurança em relação à troca de informações entre sistemas. As técnicas são a autenticação, autorização, não repudiação, integridade e privacidade.

### 4.5.1 Autenticação

Muitas vezes uma entidade, seja ela um usuário, sistema ou aplicação, necessita assegurar que outra entidade é quem ou o que ela afirma ser. A autenticação representa a existência física dessa entidade. A validação de uma entidade em particular é necessária quando se pensa em um sistema seguro. Após a autenticação, é possível decidir como será autorizada a entidade em particular. A autenticação também pode ser implementada nos *Web Services* utilizando-se de várias técnicas simples, partindo desde a autenticação utilizando usuário/senha até técnicas mais avançadas como a criptografia, a assinatura digital e/ou o certificado digital (AYALA et al., 2002).

### 4.5.2 Autorização

A autorização depende da autenticação. Depois de efetuada a autenticação, o servidor poderá autorizar o cliente a efetuar qualquer ação dependendo dos privilégios concebidos a ele. Dessa forma, é possível fazer com que entidades não autorizadas não acessem recursos ou dados não permitidos a ela. No caso de *Web Services*, essas técnicas podem ser implementadas com o uso de linguagens ou implementações como o XACL (*Extensible Access Control List*) ou o SAML (*Security Assertion Markup Language*) (AYALA et al., 2002).

### 4.5.3 Não Repudiação

A não repudiação é um serviço para assegurar que o dado enviado por um cliente chegará seguramente ao servidor. Isso gera e armazena uma comprovação da transmissão do dado. O receptor certifica a informação recebida do emissor e, desde que a comprovação da transmissão é disponível por uma terceira parte, tanto o emissor quanto o receptor não poderão negar a transmissão. Essa técnica é implementada com o uso de assinatura digital e certificado digital, mantendo a chave privada segura. A estrutura que suporta a Não Repudiação é a PKI (*Public Key Infrastructure*) e o XKMS (*XML Key Management Specification*) (ibidem).

### 4.5.4 Integridade

Conforme Ayala et al. (2002), a integridade é a autenticidade do dado ou a prova que o dado transmitido não foi alterado durante a transmissão. Para possuir uma comunicação assegurada entre um servidor e um cliente, o dado deverá estar intacto, isto é, o mesmo dado que foi enviado deverá ser recebido na outra ponta. Para manter a integridade do dado, é possível utilizar várias técnicas em vários níveis, como a implementação de algoritmos de criptografia, transformando o dado legível em dado ilegível para outros (pessoas ou sistemas) durante a transmissão. É possível utilizar técnicas como assinatura digital, certificado digital e outras técnicas de criptografia em XML.

### 4.5.5 Privacidade

A privacidade é quase relacionada com a integridade do dado. A privacidade define o cuidado com o dado privado do mundo exterior. A privacidade é obrigatória quando distribui dados

sensíveis como códigos de autorização, números de cartão de crédito e outros documentos de extrema importância (AYALA et al., 2002).

## 4.6 Considerações

Pelas comparações realizadas, o protocolo mais viável para ser utilizado nessa proposta de padrão é o SOAP, pois além de ser recomendado pelo W3C, o que garante a sua continuidade, é a possibilidade de interoperação com sistemas baseados na plataforma .NET, sendo que essa plataforma não suporta o XML-RPC. Consequentemente a escolha realizada amplia o número de linguagens de programação que poderão ser utilizadas, uma vez que o SOAP é bastante difundido.

A transmissão adotada neste *Web Service* é a transmissão de requisição e resposta, onde a empresa de rastreamento informará as posições (Latitude e Longitude) coletadas via satélite, a data e hora da coleta e as leituras efetuadas pelo sensor (temperatura e profundidade), recebendo como confirmação da transação um identificador de confirmação de registro.

No quisito segurança quatro das cinco técnicas foram contempladas: A autenticação e autorização, pois o acesso ao serviço só se dá por usuários previamente autenticados. E a integridade e privacidade, pois os dados trafegam sobre uma camada de criptografia SSL.

Para completar este *Web Service*, foi escrito um documento WSDL que descreve todo o serviço incluindo suas operações e visando facilitar a implementação dos aplicativos clientes. O documento WSDL pode ser observado no Anexo III.

# 5 RASTRO

## 5.1 Introdução

O sistema RASTRO é um sistema acessado via *web* que apresenta informações na forma de mapas de navegação ou relatórios contendo as últimas posições e os dados traçados de várias embarcações. Por trás desse sistema, encontra-se um módulo responsável pela recepção dos dados de empresas de rastreamento, que acompanham via satélite a localização das embarcações na costa brasileira. Este sistema utiliza-se de *shell scripts* e *web scripts* para executar as tarefas de captura e

processamento de dados, criação de *shapefiles* e a geração dinâmica de mapas para a *web* e relatórios (CABRAL et al., 2003).

Desde 2001 várias empresas estão certificando equipamentos que forneçam o rastreamento de posições em conformidade com o sistema RASTRO. Assim, os dados de GPS (*Global Position System*) das embarcações são repassados ao sistema que apresentará mapas e relatórios por demanda informando se a embarcação está ou não em área liberada para pesca.

O RASTRO funciona da seguinte forma: uma embarcação envia um sinal contendo os dados que deseja transmitir para o satélite, que repassa esse sinal a uma empresa de rastreamento. A empresa de rastreamento ao receber o sinal envia os dados recebidos para o proprietário da embarcação e para o local de processamento, nesse caso a Univali, onde os dados serão processados e armazenados em forma de mapas ou relatórios. O obervador, auditor ou mesmo o cliente poderão acessar essa informação pelo sistema RASTRO. A Figura 27 apresenta o diagrama de funcionamento.

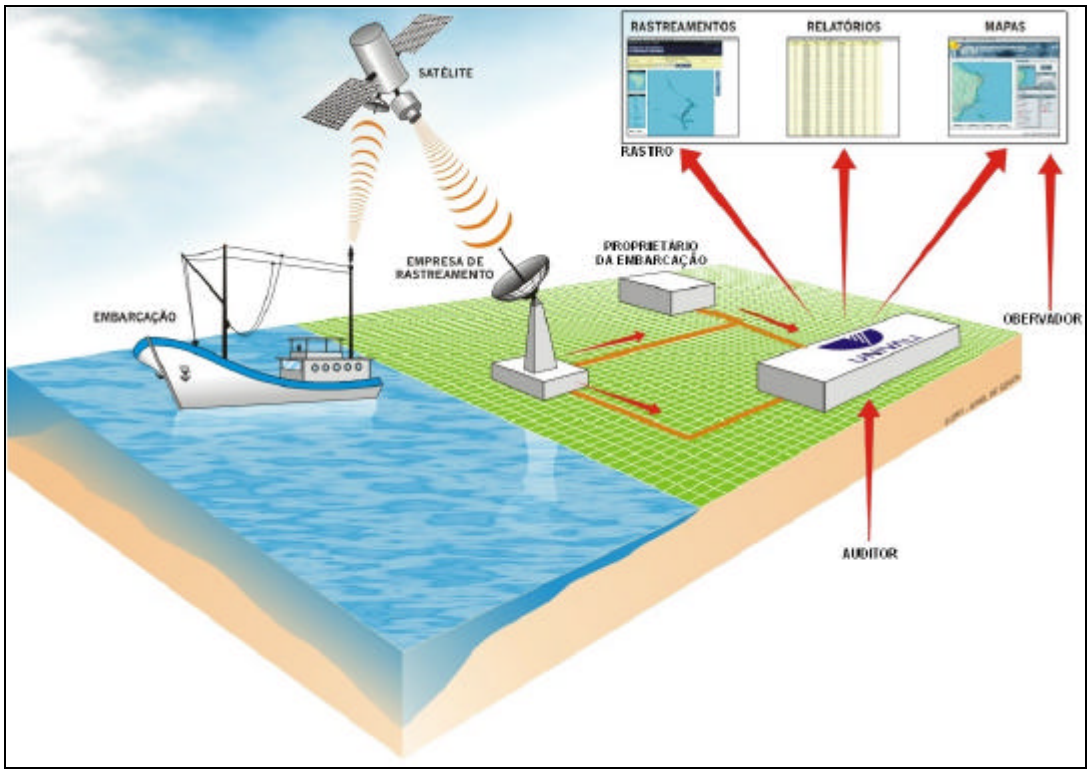



Figura 27: Arquitetura de funcionamento do sistema RASTRO.
Fonte: CABRAL et al. (2003).
Alterado: José Morelli Neto

## 5.2 Arquitetura do Sistema

As tecnologias que envolvem o sistema RASTRO, foram selecionadas seguindo dois critérios principais: a usabilidade por meio da *web* e o mundo *open source*. Fugindo um pouco desse paradigma, o Sistema de Gerenciamento de Banco de Dados (SGBD) adotado é o Oracle em conjunto com o módulo *Spatial Database Objects* (SDO). Suportando o SGBD, encontra-se o Sistema Operacional Linux, o servidor *web* Apache com suporte à linguagem de *scripts* PHP e a tecnologia MapServer. Em conjunto, essas tecnologias formam a plataforma responsável por sustentar o sistema RASTRO, o qual utiliza *scripts* em PHP para criar a *interface web* do usuário; e *shell scripts* em Linux para automatizar todo o processo.

O sistema RASTRO é estruturado em dois módulos principais: A Recepção de Dados e o Mapa de Navegação via *web* (CABRAL et al., 2003).

### 5.2.1 Recepção de Dados

A Recepção dos Dados é totalmente automatizada. Esse módulo é responsável por realizar a *interface* externa com os provedores de informações de rastreamento, que continuamente coletam o posicionamento das embarcações na costa brasileira por meio de GPS. É a partir desse ponto que os dados são entregues ao Sistema RASTRO. Nessa topologia, os equipamentos de rastreamento nas embarcações estão em contato com o sistema por intermédio das empresas de rastreamento, fornecendo, um após o outro, informações sobre o posicionamento da embarcação ao módulo de Recepção de Dados. Esse módulo processa os dados recebidos e armazena a informação de posicionamento no banco de dados Oracle com suporte espacial (ibidem).

O desafio inicial para o grupo que criou o sistema era interpretar automaticamente os *e-mails* que continham as posições do GPS. Utilizando uma ferramenta popular encontrada em distribuições Linux - o *fetchmail*, foi possível desenvolver um *shell script* que recebe e analisa os *e-mails* utilizando o protocolo POP3. Essa era a versão automatizada de "ler um *e-mail* e mapear o posicionamento da embarcação na mensagem". O *script* foi agendado para “rodar” em períodos regulares, de forma a fornecer um acesso próximo ao tempo real da informação via *web*.

Atualmente, muitas outras formas de recepção de dados e recuperação estão em funcionamento. Padrões legados foram estudados e implementados para também certificar equipamentos de terceiros, incluindo o acesso aos dados GPS via Telnet, entre outros (CABRAL et al., 2003).

### 5.2.2 Mapa de navegação via Web

O Mapa de navegação via *Web* é uma *interface* intuitiva de interação com o usuário que apresenta camadas contento às últimas posições (Figura 28) e os pontos rastreados em um mapa base (Figura 29). As funções de *interface* são disponíveis conforme o perfil do usuário e a sua permissão de acesso ao sistema.


Figura 28: Navegação em mapas via Web – últimas posições.

Segundo Cabral et al. (2003), o RASTRO permite a um usuário possuir um dos seguintes perfis:

?? **Auditor:** O auditor pode acessar toda a informação e efetuar consultas regionais de qualquer intervalo de tempo (*e.g.* observe a definição de tempo na Figura 29). Além disso,

é permitido, a esse perfil, gerar um relatório no formato PDF para documentar oficialmente a atividade de pesca como definido pela lei brasileira.

?? **Observador:** O observador de bordo está apto a acessar toda a informação, restrito apenas aos últimos dez dias. Essa informação ajuda o coordenador da equipe a principalmente acompanhar os pontos de partida e chegada de cada embarcação;

?? **Companhia:** Esse perfil possui permissão de acesso apenas às informações restritas das embarcações de uma determinada companhia, mas não é obrigado a executar consultas regionais de qualquer intervalo de tempo. A disponibilidade dos dados na *web*, ajuda aos funcionários da companhia a coordenar operações de qualquer lugar e a qualquer momento.

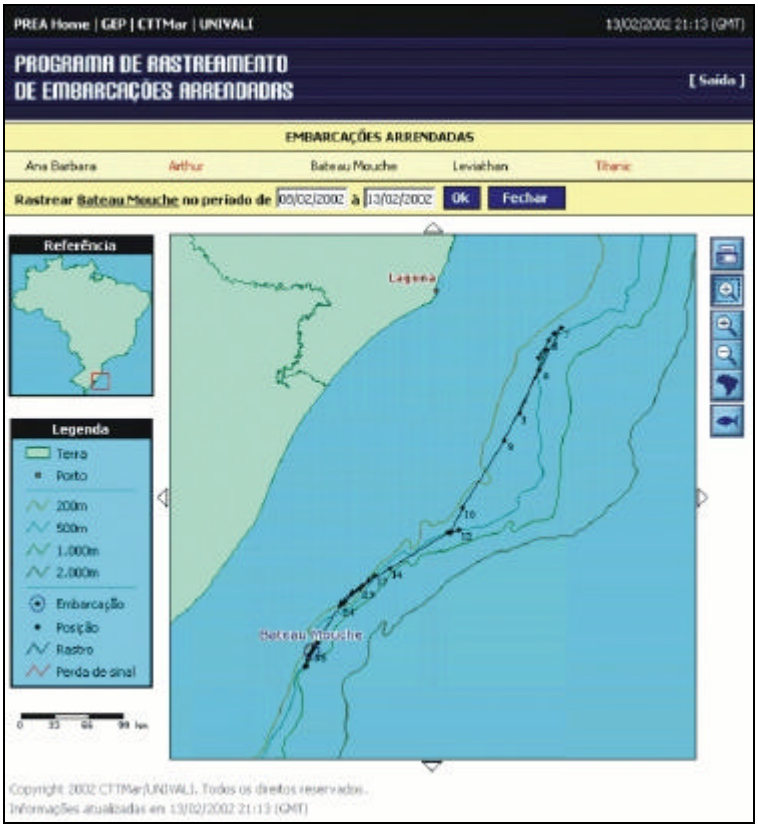



Figura 29: Pontos rastreados de uma embarcação.

A visão de últimas posições resume a distribuição de todas as embarcações ao longo da costa brasileira. É apresentado em preto o ponteiro de coordenadas (rastro) marcando o nome das embarcações que tiveram sua localização bem sucessida dentro das últimas 4 horas. Por outro lado,

embarcações que estavam fora de alcance por mais de 4 horas são destacadas em vermelho com um "X", apresentando sua última posição conhecida.

A visão de pontos traçados é utilizada para monitorar a atividade de pesca, apresentando todas as posições recebidas dentro de um intervalo de tempo. Cada posição marcada está conectada com a posição seguinte por uma linha preta, a não ser que um sinal de "não encontrado" seja recebido entre duas posições - nesse caso, uma linha vermelha irá ser desenhada. Se possível, ao longo de cada marca de posição é apresentado um número que identifica os dados da posição apresentados abaixo do mapa (CABRAL et al., 2003).

Quando uma embarcação ocupa a área de pesca, a velocidade estimada entre os pontos é reduzida para 4 nós. Um botão especial na *interface*, permite ao usuário observar apenas as atividades de baixa velocidade; nitidamente, marcando o espaço que provavelmente a embarcação estava em atividade de pesca. Essa informação é essencial ao apresentar relatórios oficiais ao governo para confirmar que a operação está “correndo” de acordo com as políticas de gerenciamento de pesca.

Outras características normalmente encontradas em aplicações *webgis* também estão disponíveis no RASTRO, como recursos de *zoom in* e *zoom out*, zerar a visualização e consulta de dados (ibidem).

## 5.3 Considerações

Uma das finalidades deste trabalho é substituir o módulo de Recepção de Dados inicialmente utilizado no sistema RASTRO por uma implementação do padrão proposto, permitindo ao sistema, receber dados de qualquer empresa de rastreamento que deseje entrar em conformidade com o padrão proposto.

Isso vem de encontro com um problema comum entre os clientes de serviços de rastreamento, pois não é possivel optar pelo *software* de monitoramento de uma empresa e o serviço de rastreamento de outra, ou seja, cada empresa de rastreamento fornece junto com a entrega do sinal um *software* proprietário. Uma vez que esse padrão seja adotado pelas empresas de

rastreamento, se for conveniente, o cliente poderá optar pelo serviço de uma empresa, e o *software* de outra.

# III - DESENVOLVIMENTO

## 1 INTRODUÇÃO

Para apresentar os resultados obtidos no estudo sobre os *Web Services*, bem como o estudo do protocolo SOAP, foram implementados dois sistemas para apresentar o padrão proposto. O primeiro é um protótipo descrito na Seção 4.3.1 que apresenta, na íntegra, o uso de todos os recursos fornecidos pelo padrão, como, por exemplo, o uso do modelo visando o transporte de informações coletadas de sensores. O segundo, apresentado na Seção 4.3.2 foi concebido para atender às necessidades do sistema RASTRO, contendo um número reduzido de dados suportados. Informações sobre o padrão proposto podem ser encontrado no capítulo 3 – Descrição do Padrão Proposto. Para a elaboração desse desenvolvimento, foram seguidos os seguintes passos: Modelagem do Sistema, Descrição, Implementação e Validação do Padrão Proposto, baseando-se em um protótipo e uma implementação para permitir ao sistema RASTRO usufruir das vantagens desta proposta.

## 2 MODELAGEM DO SISTEMA

A modelagem do sistema tem como intuito apresentar os Modelos de Caso de Uso e Modelo de Entidade e Relacionamento (MER), que apresenta os aspectos lógicos e de projeto do padrão proposto. Esses modelos demonstram de forma simples o funcionamento de um sistema compatível com o padrão proposto.

### 2.1 Diagrama de Caso de Uso

Segundo Furlan (1998), os diagramas de caso de uso se apresentam como um modelo para descrever um panorama externo do sistema apresentando suas interações com o mundo exterior. Os diagramas de caso de uso apresentam uma visão macro do sistema, representando uma visão de alto nível de funcionalidade, mediante os diferentes tipos de requisições dos usuários.

A Figura 30 apresenta o diagrama de caso de uso do padrão proposto. O Cliente é uma aplicação que fará uso de chamadas via *Web Service* aos métodos disponíveis remotamente no

servidor, que também é um *Web Service*. Cada método de um *Web Service* asemelha-se a um procedimento em uma linguagem de programação qualquer. Ao invocar um método, deve ser passado alguns parâmetros que serão utilizados pelo servidor. Após finalizado o processamento, o servidor retornará a resposta solicitada.

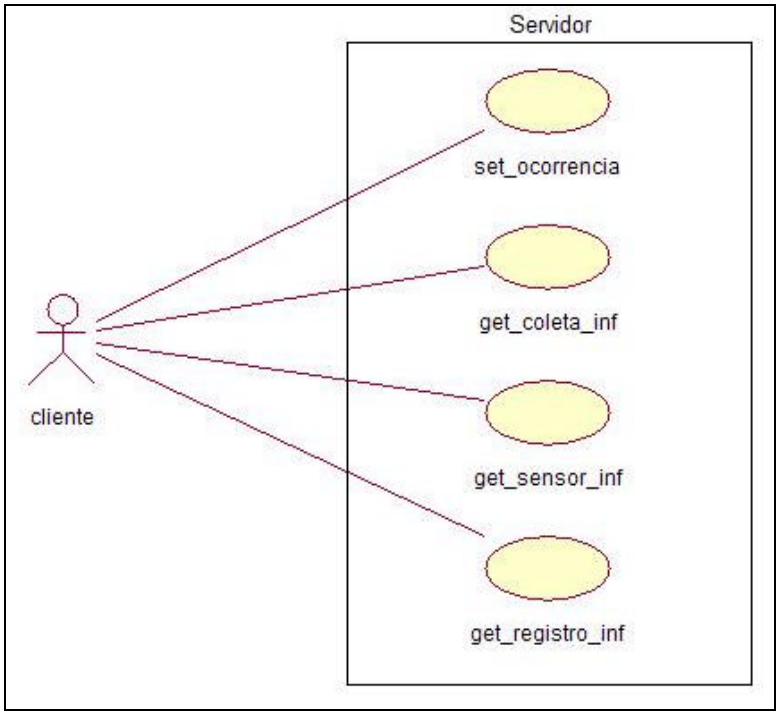



Figura 30: Diagrama de Caso do padrão proposto.

As Figuras 31, 32, 33 e 34 , descritas a seguir, apresentam os métodos e seus requisitos funcionais de forma detalhada.

| Use Case : set_ocorrencia |
|---|
| Quem dispara: Aplicação Cliente |
| Curso Básico: A aplicação cliente registrará uma ocorrência passando os dados para esse método. Como resposta, receberá um número de protocolo que se refere à confirmação do cadastro dos dados. O número de protocolo é o índice de uma tabela de *log*, gerada pelo sistema confirmando as operações realizadas. |
| Cursos alternativos/secundários: não possui |
| Pré-condições: A mensagem SOAP de requisição deverá possuir em seu cabeçalho o código e a senha do cliente para efetuar a sua identificação no sistema. |
| Pós-condições: não possui |

Figura 31: Caso de Uso Registrar Ocorrência.

| Use Case : get_coleta_inf |
|---|
| Quem dispara: Aplicação Cliente |
| Curso Básico: A aplicação cliente envia uma mensagem SOAP solicitando a execução desse método e passando o código do cliente. Receberá como resposta todos os pontos de coleta disponíveis para esse cliente. Durante a execução desse método é adicionado um registro em uma tabela de *log* indicando informações referentes à execução do método. |
| Cursos alternativos/secundários: não possui |
| Pré-condições: A mensagem SOAP de requisição deverá possuir em seu cabeçalho o código e a senha do cliente para efetuar a sua identificação no sistema. |
| Pós-condições: não possui |

Figura 32: Caso de Uso Solicitar Pontos de Coleta.

| Use Case : get_sensor_inf |
|---|
| Quem dispara: Aplicação Cliente |
| Curso Básico: A aplicação cliente envia uma mensagem SOAP solicitando a execução desse método e passando o código do ponto de coleta. Receberá como resposta todos os sensores disponíveis para esse ponto de coleta. Durante a execução desse método, é adicionado um registro em uma tabela de *log* indicando informações referentes a execução do método. |
| Cursos alternativos/secundários: não possui |
| Pré-condições: A mensagem SOAP de requisição deverá possuir em seu cabeçalho o código e a senha do cliente para efetuar a sua identificação no sistema. |
| Pós-condições: não possui |

Figura 33: Caso de Uso Solicitar Sensores.

| Use Case : get_ocorrencia_inf |
|---|
| Quem dispara: Aplicação Cliente |
| Curso Básico: A aplicação cliente envia uma mensagem SOAP solicitando a execução desse método e passando o código do ponto de coleta. Receberá como resposta todos os registros para esse ponto de coleta com seus respectivos sensores (se houver) dos últimos 10 dias. Durante a execução desse método, é adicionado um registro em uma tabela de *log* indicando informações referentes a execução do método. |
| Cursos alternativos/secundários: não possui |
| Pré-condições: A mensagem SOAP de requisição deverá possuir em seu cabeçalho o código e a senha do cliente para efetuar a sua identificação no sistema. |
| Pós-condições: não possui |

Figura 34: Caso de Uso Solicitar Ocorrências.

Os dados serão consultados ou gravados a partir de tabelas em um banco de dados. A seguir, será apresentado o Modelo de Informação utilizado nesse projeto.

## 2.2 Modelo de Informação

Para modelar os dados que serão transportados em sistemas de telemetria por meio do *Web Service* proposto, utilizou-se o Modelo Entidade-Relacionamento (MER), dividido em modelo lógico (conceitual) e modelo físico, conforme apresentados nas Figuras 35 e 36, respectivamente.

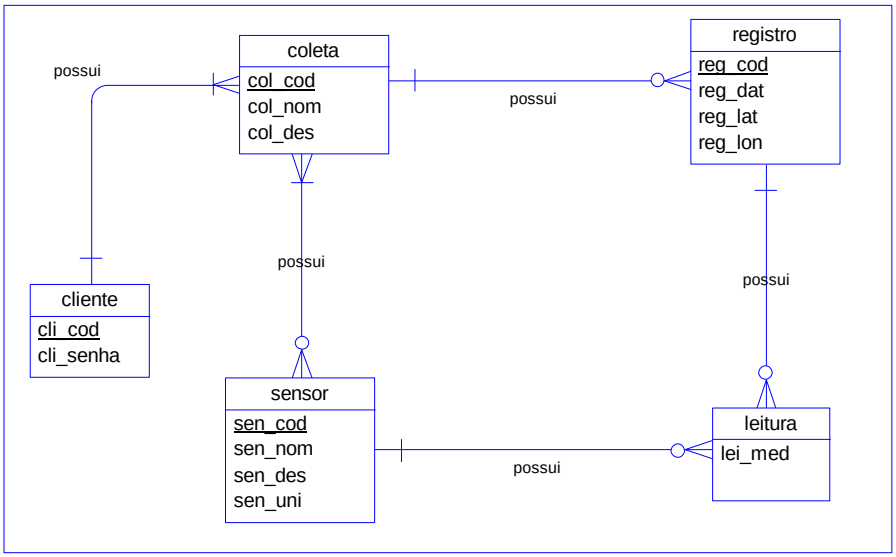



Figura 35: Modelo ER lógico dos dados.

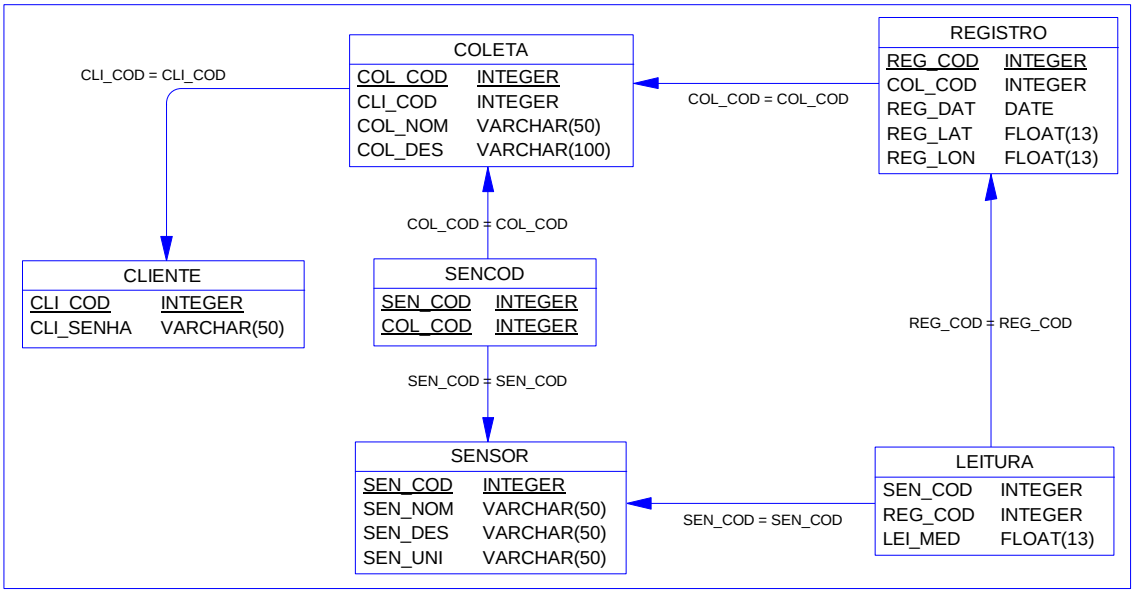



Figura 36: Modelo ER físico dos dados.

O Dicionário de Dados é formado pela descrição das tabelas utilizadas no banco de dados proposto para o padrão. A Tabela 3 apresenta todas as tabelas utilizadas no sistema, enquanto as

Tabelas 4, 5, 6, 7 , 8 e 9 descrevem individualmente cada tabela de dados utilizada. Toda a descrição apresentada a seguir é baseada no modelo físico apresentado na Figura 36.

Tabela 3: Descrição das tabelas de dados utilizadas no sistema

| Nome da Tabela | Descrição |
|---|---|
| CLIENTE | Tabela que armazena as informações necessárias para que o cliente possa se autenticar ao utilizar os métodos. |
| COLETA | Possui informações referentes aos pontos de coleta, que pode ser por exemplo, uma embarcação ou veículo. Essa tabela lista os pontos de coleta de um determinado cliente. |
| SENSOR | Armazena uma lista de sensores que podem estar ou não associados a um ou mais pontos de coleta. |
| SENCOD | Tabela que associa os sensores presentes em determinados pontos de coleta. |
| REGISTRO | Essa tabela armazena os registros informados pelos clientes. É a informação básica para que um ponto de coleta possa utilizar o sistema. |
| LEITURA | Armazena as informações referentes a leituras realizadas pelos sensores. Essa tabela armazena apenas os registros referentes aos pontos de coleta que possuem sensores. |

Tabela 4: Dicionário de Dados da tabela Cliente

| Nome do Atributo | Tipo/Tamanho | Descrição |
|---|---|---|
| cli_cod | Inteiro | Código do Cliente – Chave Primária. |
| cli_senha | Alfanumérico/30 | Senha do cliente para efetuar a autenticação no sistema. |

Tabela 5: Dicionário de Dados da tabela Coleta

| Nome do Atributo | Tipo/Tamanho | Descrição |
|---|---|---|
| col_cod | Inteiro | Código do ponto de Coleta – Chave Primária. |
| col_nom | Alfanumérico/50 | Nome do ponto de coleta. |
| col_des | Alfanumérico/100 | Descrição do ponto de coleta. |
| cli_cod | Inteiro | Código do Cliente – Chave Estrangeira do CLIENTE |

Tabela 6: Dicionário de Dados da tabela Sensor

| Nome do Atributo | Tipo/Tamanho | Descrição |
|---|---|---|
| sen_cod | Inteiro | Código do Sensor – Chave Primária. |
| sen_nom | Alfanumérico/50 | Nome do Sensor. |
| sen_des | Alfanumérico/50 | Descrição do sensor. |
| sen_uni | Alfanumérico/50 | Unidade de medida que o sensor está coletando. |

Tabela 7: Dicionário de Dados da tabela SenCod

| Nome do Atributo | Tipo/Tamanho | Descrição |
|---|---|---|
| sen_cod | Inteiro | Código do Sensor – Chave Estrangeira do SENSOR. |
| col_cod | Inteiro | Código do ponto de Coleta – Chave Estrangeira da COLETA. |

Tabela 8: Dicionário de Dados da tabela Registro

| Nome do Atributo | Tipo/Tamanho | Descrição |
|---|---|---|
| reg_cod | Inteiro | Código do Registro – Chave Primária. |
| reg_dat | DataHora | Data e hora de coleta do registro. |
| reg_lat | Real/13 | Latitude no momento da coleta. |
| reg_lon | Real/13 | Longitude no momento da coleta. |
| col_cod | Inteiro | Código do ponto de Coleta – Chave Estrangeira da COLETA. |

Tabela 9: Dicionário de Dados da tabela Leitura

| Nome do Atributo | Tipo/Tamanho | Descrição |
|---|---|---|
| sen_cod | Inteiro | Código do Sensor – Chave Estrangeira de SENSOR. |
| reg_cod | Inteiro | Código do Registro – Chave Estrangeira de COLETA. |
| lei_med | Real/13 | Leitura medida no sensor sen_cod. |

# 3 DESCRIÇÃO DO PADRÃO PROPOSTO

O padrão proposto é apresentado nesse trabalho na forma de um documento em anexo. O intuito é facilitar a divisão entre o trabalho de graduação e o padrão em si. Assim sendo, o Anexo IV apresenta os métodos e tipos de dados que devem ser utilizados para seguir essa proposta. O

modelo de informação apresentado na Figura 35 (Página 64) refere-se aos tipos de dados utilizados no padrão proposto, bem como o modelo utilizado para a elaboração do protótipo apresentado na Seção 4.3.1. Essa proposta foi elaborada baseando-se no item 2.7 - Construção de Padrões.

# 4 IMPLEMENTAÇÃO DO PADRÃO PROPOSTO

## 4.1 Introdução

Este capítulo abordará as tecnologias utilizadas para a implementação do padrão proposto, uma descrição geral da implementação realizada para o sistema RASTRO, assim como o protótipo inicialmente desenvolvido.

## 4.2 Tecnologia

Entre as tecnologias utilizadas para o desenvolvimento do protótipo e do sistema pode-se citar o uso do PHP, da biblioteca PEAR, do MySQL e do Oracle. A seguir será dada uma visão geral de cada uma das tecnologias envolvidas.

### 4.2.1 PHP

A sigla PHP é um acrônico para "PHP: *Hypertext Preprocessor*", sendo uma linguagem de *scripts* de uso geral muito utilizada e especialmente elaborada para o desenvolvimento de aplicações *web* embutível dentro do HTML (PHP GROUP, 2003a).

O PHP foi escolhido para a implementação tanto do protótipo, quanto do sistema que permitirá ao RASTRO receber informações por meio de um *Web Serivce*, pois o próprio sistema RASTRO foi implementado utilizando essa linguagem. Pode-se citar também que o PHP é uma linguagem *Open Source*, isto é, ela possui seu código fonte disponível para *download*, além de ser livre para uso em qualquer finalidade.

### 4.2.2 PEAR

A sigla PEAR deriva de "PHP *Extension and Application Repository*" e seu propósito é fornecer uma biblioteca estruturada de códigos PHP *Open Source* de usuários, elaborados de forma padronizada e com um sistema de empacotamento dividido por implementações, para facilitar a manutenção e distribuição da biblioteca (PHP GROUP, 2003b).

Entre os pacotes fornecidos pelo PEAR, encontra-se um que permitiu o uso da linguagem PHP para a implementação do *Web Service* proposto: o pacote SOAP. A versão utilizada para a implementação desse projeto foi a de número 0.8 rc2, que ainda apresenta-se como uma versão beta. A documentação das funções fornecidas por esse pacote ainda não foi elaborada, sendo assim, para o desenvolvimento deste trabalho, foi necessário o estudo de muitos exemplos já implementados, além de várias consultas ao serviço de *News* do PHP.

### 4.2.3 MySQL

O MySQL é um SGBD (Sistema de Gerenciamento de Banco de Dados) Relacional que utiliza a linguagem padrão SQL (*Structured Query Language)* e é largamente utilizado em aplicações *web*. O MySQL é o mais popular SGBD *Open Source*, isto é, ele pode ser distribuido livremente, e possui seu código fonte disponível para *download* (MYSQL, 2003).

O Mysql foi escolhido para o desenvolvimento do protótipo por ser um SGBD simples e leve (tanto para iniciar, quanto para utilizá-lo em um equipamento *desktop*), simples e de fácil utilização.

### 4.2.4 Oracle

O Oracle é um SGBD (Sistema de Gerenciamento de Banco de Dados) Relacional e Distribuído, ou seja, um conjunto de sistemas de banco de dados conectados através de uma rede de comunicações. Desta forma, um usuário em qualquer sistema de banco de dados pode ter acesso a qualquer dado na rede, como se este dado estivesse no próprio sistema de banco de dados do usuário (CERÍCOLA, 1995). Seu principal objetivo é fornecer um ambiente adequado e eficiente

para recuperação e armazenamento da informação (SILBERSCHATZ, KORTH & SUDARSHAN, 1999).

Conforme Cerícola (1995), o Oracle apresenta vantagens como, por exemplo, simplicidade e uniformidade, independência total dos dados, *interfaces* de alto nível para usuários finais, melhoria na segurança dos dados, custo de comunicações mais baixo, tempo de resposta mais rápido e melhoria na confiabilidade.

Por apresentar estas importantes características e vantagens, o Oracle é o mais conhecido e difundido no mercado, sendo composto por um grupo de ferramentas bastante amplo e completo e possuindo portabilidade com várias marcas como: MS-DOS, Unix, Macintosh, Siemens, HP e OS/2 (CERÍCOLA, 1995).

O Oracle foi utilizado nessa implementação por ser o SGBD empregado no Sistema RASTRO. Já o RASTRO utiliza o Oracle devido a sua capacidate de armazenamento de dados espaciais.

## 4.3 Validação do Padrão Proposto

### 4.3.1 Protótipo

Para exemplificar melhor o padrão proposto, foi elaborado um protótipo que possui todas as especificações definidas. Esse protótipo possui a entrada de dados de forma manual (o serviço em si é executado sem interação com o usuário) e foi elaborado utilizando a linguagem PHP e o Banco de Dados MySQL em conjunto com a biblioteca PEAR::SOAP. Essas tecnologias foram abordadas anteriormente neste capítulo.

Esse protótipo, aparentemente simples, apresenta recursos avançados, e muitas vezes de difícil compreensão. Um dos fatores mais relevantes na dificuldade da implementação foi a falta de documentação explicando de forma prática a implementação. A ausência de documentação deriva do fato de que a biblioteca ainda encontra-se em fase de desenvolvimento, não possuindo uma documentação formal. O material utilizado foi baseado em exemplos já existentes (embora muitos

exemplos não funcionassem de acordo) e extensivas consultas ao Fórum de desenvolvedores da biblioteca SOAP.

Superados estes obstáculos, foi possível concluir este protótipo totalmente funcional de acordo com o padrão proposto. A implementação baseou-se no Modelo de Informação apresentado anteriormente na Figura 36. O Banco de Dados MySQL foi gerado automaticamente pela Ferramenta *Powerdesigner DataArchitect* versão 9.5.

A implementação foi dividida em dois módulos principais: o módulo servidor do *Web Service* e o módulo cliente. A seguir serão apresentadas algumas particularidades de cada módulo.

#### 4.3.1.1 Módulo Servidor

O módulo servidor é o responsável por receber as requisições SOAP dos clientes, processá-las de acordo com os métodos e retornar os resultados. É esse módulo que interage com o Banco de Dados onde encontram-se todas as informações registradas, e foi implementado utilizando todos os métodos propostos. O servidor possui apenas uma *interface* com o usuário que apresenta o arquivo WSDL responsável por descrever o serviço, e que pode ser visualizado no Anexo III.

#### 4.3.1.2 Módulo Cliente

O módulo cliente está dividido em três telas principais, que serão apresentadas e descritas a seguir. A Figura 37 apresenta a tela inicial do protótipo. Nela existem duas opções de uso: Inserir Ocorrência e Consultar Registros e Sensores.

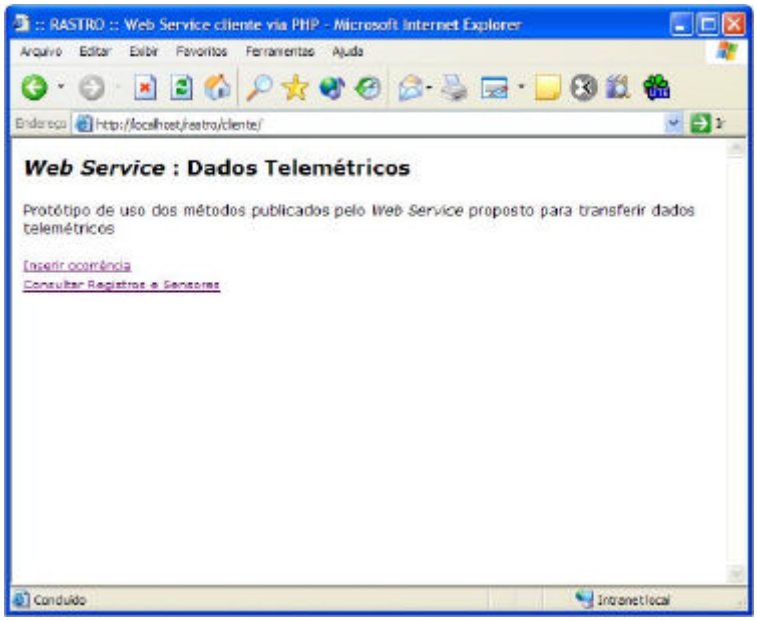



Figura 37: Tela inicial do Módulo Cliente.

A Figura 38 apresenta a tela responsável por inserir ocorrências no sistema. Ao entrar nessa tela serão apresentados inicialmente os campos Cliente e Senha. Após preenchidos corretamente esses campos, será executado um método via *Web Service* que retornará todos os pontos de coleta pertencentes ao cliente informado.

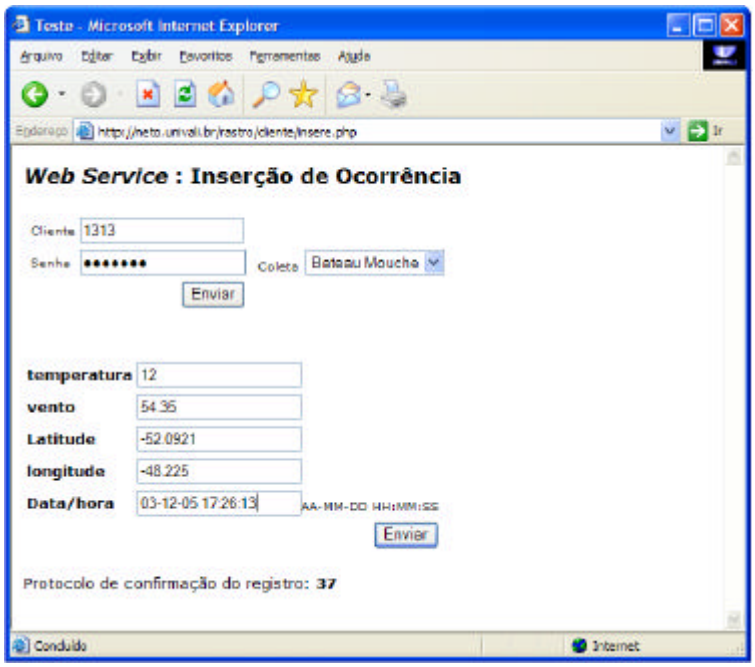



Figura 38: Tela de inserção de ocorrências.

Após selecionado um dos pontos de coleta disponíveis será automaticamente executado outro método, que retornará todos os sensores da embarcação selecionada, mais a latitude, longitude e um campo para acrescentar a data/hora. Em seguida, deverão ser preenchidos os valores dos sensores, latitude e longitude, e pressionado o botão enviar. Nesse momento, será executado o método responsável por cadastrar uma ocorrência, que enviará para o *Web Service* todos os dados preenchidos e retornará um número de protocolo de confirmação do registro. Essa tela implementa três dos quatro métodos propostos: Registrar Ocorrência, Solicitar Pontos de Coleta e Solicitar Sensores.

Por fim, a Figura 39 apresenta a tela responsável apenas por consultas, e ela implementa o último método proposto: o Solicitar Ocorrências. Semelhante a tela anterior, essa também faz-se necessário informar o Cliente e a Senha. Após pressionado o botão Enviar, serão retornados todos os Pontos de Coleta disponíveis para determinado cliente. A diferença básica é a existência de dois *Checkbox* por onde é possível escolher quais os métodos que serão executados. Nesse exemplo, foram selecionados os dois *checkbox*, o que retornou os sensores do ponto de coletas escolhido e os registros dos últimos 10 dias desse ponto de coletas.

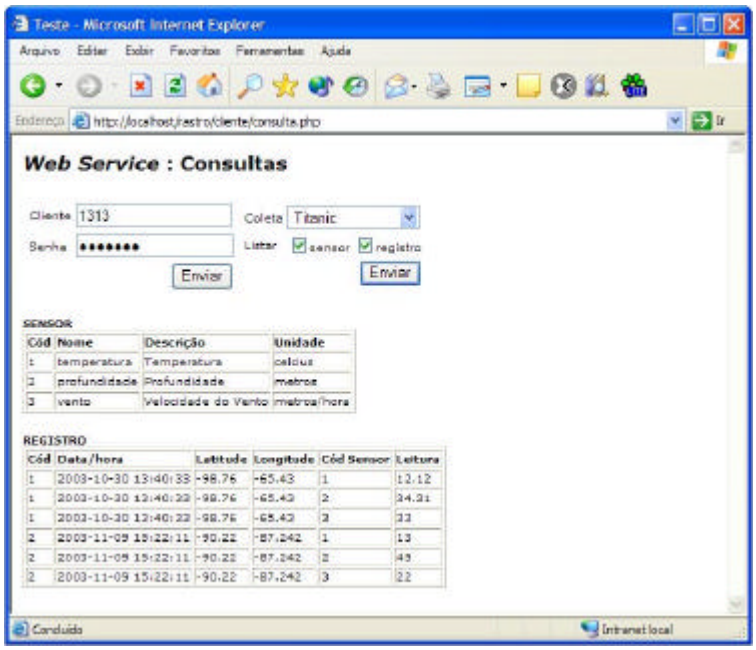



Figura 39: Tela de consultas do Módulo Cliente.

### 4.3.2 Implementação do Padrão no Sistema RASTRO

O sistema RASTRO, como apresentado no Capítulo 5, é dividido em dois módulos: Recepção de Dados e Mapa de navegação via *web*. O funcionamento incial seguia as seguintes etapas:

?? A cada 15 minutos o agendador de tarefas do Linux, o *crontab*, executa um *shell script* que dispara a execução do *fetchmail*, responsável por conectar no servidor de *e-mail* via POP3 e copiar as mensagens para repassá-las a outro programa: chamado MDA (*Mail Delivery Agent*);

?? O MDA analisa todas as mensagens baixadas, uma a uma, coletando de cada mensagem informações como o posicionamento (Latitude e Longitude) e o nome da embarcação que enviou a informação.

?? De acordo com os parametros passados na execução do programa, o MDA grava os registros ou em um arquivo texto, ou diretamento no Banco de Dados Oracle.

A Figura 40 ilustra o funcionamento do módulo de Recepção de Dados do sistema RASTRO antes da implementação proposta.

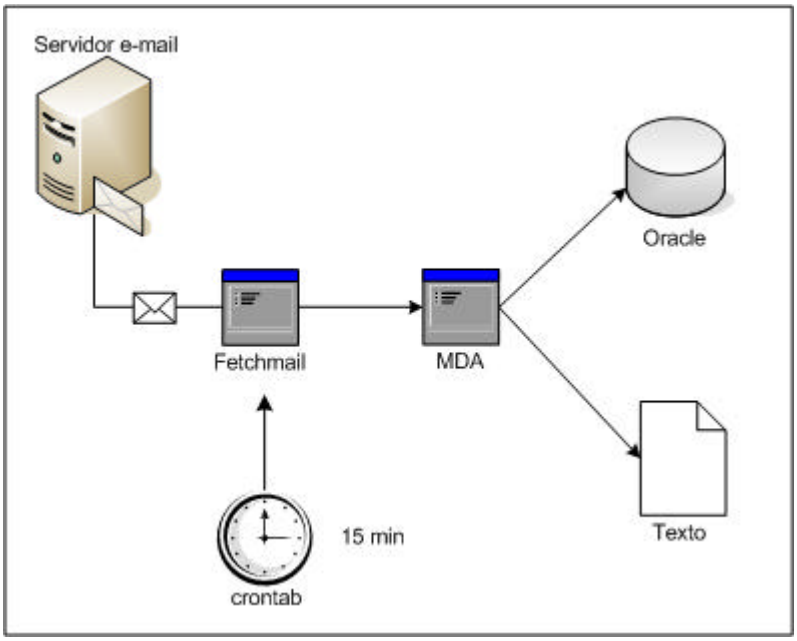



Figura 40: Funcionamento inicial do módulo de Recepção de Dados.

De forma inversa ao protótipo apresentado anteriormente, a implementação no sistema RASTRO não necessita de nenhuma interação humana – os dados são recebidos por e-mail, e todo o processo é automático. Para permitir ao RASTRO usufruir dos recursos do *Web Service* proposto e para validar esse modelo, a implementação realizada substitui o programa MDA por duas aplicações *Web Services*: um cliente e um servidor.

A aplicação cliente é responsável por se conectar ao servidor POP3 (*e-mail*) a cada 15 minutos, baixando as mensagens uma a uma, analisando-as e colentado o posicionamento (latitude e longitude), data e hora de coleta e o nome da embarcação. Caso não exista uma posição válida, o valor *null* será repassado como latitude e longitude. Em seguida uma operação via *Web Service* é executada para consultar se a embarcação está ou não cadastrada no banco de dados do RASTRO. Caso esteja as informações são repasadas via *Web Service* para a aplicação servidor, que então armazena os dados no banco de dados Oracle ou em um arquivo texto. No caso da embarcação não existir no RASTRO, a aplicação cliente salvará as informações em um arquivo de *Log* para constar a tentativa de transmissão.

A Figura 41 apresenta a implementação realizada no módulo de Recepção de Dados, para compatibilizar o sistema RASTRO ao padrão proposto.

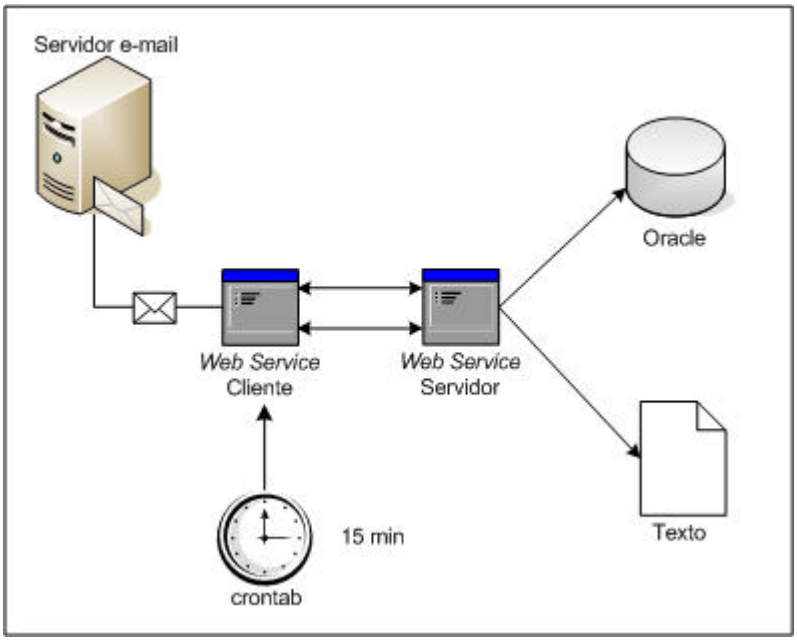



Figura 41: Alteração efetuada no módulo de Recepção de Dados.

Devido ao fato do sistema RASTRO ainda não suportar o armazenamento de valores coletados a partir de sensores, a implementação realizada limita-se apenas a aceitar informações referentes ao posicionamento e ponto de coleta.

# IV - CONCLUSÕES E RECOMENDAÇÕES

Foram apresentados nesse trabalho vários conceitos permitindo o embasamento necessário para a elaboração de uma proposta visando padronizar o transporte de dados telemétricos. Ao findar esse trabalho, pode-se obter um documento que aborda entre seus capítulos, informações sobre Telemetria, Padrões e *Web Services.*

Em termos de Telemetria, os estudos foram voltados à apresentação do conceito, exemplos de classificação e algumas vantagens e desvantagens. O objetivo desse capítulo foi apenas a apresentação conceitual do tema, uma vez que esse projeto não visa propor alterações na forma de comunicação existente hoje, e sim apresentar os benefícios do uso de *Web Services* para o transporte desses dados no nível de aplicação.

Uma vez delineado os estudos, percebeu-se a necessidade de apresentar uma proposta de padrão que exemplifique e apresente de forma minuciosa o objetivo do projeto. Para isso, foi necessário efetuar uma revisão bibliográfica que contemple o tema Padrões, apresentando a sua importância, definição, algumas entidades de Normalização, exemplos e uma introdução de como se constrói um padrão.

Em seguida, é apresentada uma seção que aborda os *Web Services*, descrevendo seu funcionamento, vantagens de utilização, e alguns conceitos acerca desse tema, como XML, UDDI e WSDL – uma linguagem elaborada para descrever um serviço. Existem dois protocolos distintos utilizados para a implementação de um *Web Service*: SOAP e XML-RPC. Ambos são apresentados de forma profunda, e por fim é feito um comparativo entre os dois permitindo a escolha do mais viável para o projeto.

Como estudo de caso foi utilizado um sistema de monitoramento de embarcações - O sistema RASTRO. Foi dedicado um capítulo a esse sistema para permitir um melhor entendimento dos processos de um sistema de monitoramento. Uma das finalidades desse trabalho foi substituir um módulo do sistema RASTRO para adequá-lo ao padrão proposto.

Após finalizado o levantamento bibliográfico, iniciou-se o processo de desenvolvimento, onde foi elaborado a proposta de padrão nos moldes de uma RFC (*Request for Comment*) utilizando *Web Services*, que foi empregado para a elaboração de um protótipo de testes utilizado para apresentar todas as funcionalidades do modelo proposto e, em seguida, foi alterado um dos módulos do sistema RASTRO, permitindo a ele usufruir dessa proposta.

Este trabalho ajuda a contornar um problema comum entre as empresas de rastreamento e seus clientes, pois inicialmente não é possível optar pelo *software* de monitoramento de uma empresa e o serviço de rastreamento de outra. Uma vez padronizado esse canal de dados, um cliente poderá possuir apenas um sistema de monitoramento desenvolvido por qualquer empresa e utilizar o serviço de rastreamento de uma ou várias empresas de rastreamento diferentes.

Sendo o objetivo deste trabalho atender a necessidade de padronização no intercâmbio de informações de telemetria entre sistemas automatizados em nível de aplicação, foi estabelecido um padrão na forma de um documento, abordando as operações propostas e os tipos de dados utilizados. A descrição é apresentada na forma de um documento WSDL, que apresenta o serviço. Os dados são transacionados por meio do protocolo SOAP sob o HTTPS (*HyperText Transfer Protocol Secure*), o que garante a criptografia dos dados transportados.

É importante ressaltar a distinção deste tema enquanto um Trabalho de Conclusão de Curso de Ciência da Computação, ao considerar a inovação no tratamento de um problema não através da produção de um sistema, mas sim por meio de uma padronização das transações entre sistemas, ou seja, na criação de um protocolo de transações de dados de telemetria em camada de aplicação.

Como recomendação para o sistema RASTRO, pode-se citar a implementação de um suporte a coletas de valores de sensores nas embarcações, a codificação de programas "exemplos" que poderão ser repassados a empresas de rastreamento que desejam se adequar ao sistema, e a remoção por completo do recebimento de dados telemétricos por meio de *e-mail*, utilizando apenas *Web Services* para esse fim.

Já para o padrão proposto incorporado nesse trabalho na forma de um anexo, é recomendado a definição de novas operações permitindo maior flexibilidade inter-sistemas, e o uso de estruturas

de PKI (*Public Key Infrastructure*), XKMS (*XML Key Management Specification*) para assegurar as transações efetuadas entre empresas de rastreamento e seus clientes.

# BIBLIOGRAFIA


ABNT - Associação Brasileira de Normas Técnicas. **Newsletter Virtual**. Ano 2, n. 14, set. 2000. Disponível em: <http://www.abnt.org.br/newsletter/edicao14/body.htm>. Acesso em: 04 out. 2003.

ABNT - Associação Brasileira de Normas Técnicas. **O que é Normalização**. 2003a. Disponível em: <http://www.abnt.org.br/normal_oque.htm>. Acesso em: 02 out. 2003.

ABNT - Associação Brasileira de Normas Técnicas. **Apresentação**. 2003b. Disponível em: <http://www.abnt.org.br/instit_apresen.htm>. Acesso em: 02 out. 2003.

AMN - Asociación Mercosur de Normalización. **Informações Gerais**. 2003. Disponível em: <http://www.amn.org.br/br/00401007.asp >. Acesso em: 03 out. 2003.

ANEEL - AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA. **Estudo de Informações Hidrológicas - Telemetria**. Disponível em: <http://www.aneel.gov.br/area.cfm?id_area=129>. Acesso em: 08 mar. 2003.

ANSI - American National Standards Institute. **Frequently Asked Questions**. 2003. Disponível em: <http://www.ansi.org/about_ansi/faqs/faqs.aspx?menuid=1>. Acesso em: 02 out. 2003.

APPLE Computers. **XML-RPC vs. SOAP**. Disponível em: <http://developer.apple.com/techpubs/macosx/Networking/WebServices/1_intro_folder/chapter_1_section_5.html>. Acesso em: 12 jun. 2003.

AUSTIN, Daniel (Ed.) et al. **Web Services Architecture Requirements**. 2002. Disponível em: <http://www.w3.org/TR/wsa-reqs>. Acesso em: 05 mai. 2003.

AYALA, Dietrich et al. **Professional Open Source Web Services**. UK: Wrox Press Ltd, 2002. ISBN 1-861007-46-9.

BELLWOOD, Tom et al. **UDDI Version 3.0**. 2002. Disponível em: <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>. Acesso em: 20 mai. 2003.

BIRON, Paul V.; MALHOTRA, Ashok. **XML Schema Part 2: Datatypes**. 2001. Disponível em: <http://www.w3.org/TR/xmlschema-2/>. Acesso em: 06 jun. 2003.

BOS, Bert**. What is a good standard? An essay on W3C's design principles**. 2003. Disponível em: < http://www.w3.org/People/Bos/DesignGuide/committee.html#Design>. Acesso em: 14 dez. 2003.

BOX, Don et al. **Simple Object Access Protocol (SOAP) 1.1**. 2000. Disponível em: <http://www.w3.org/TR/SOAP>. Acesso em: 10 mai. 2003.

BRAY, Tim (Ed.); HOLLANDER, Dave (Ed.); LAYMAN, Andrew (Ed.). **Namespaces in XML**. 1999. Disponível em: <http://www.w3.org/TR/1999/REC-xml-names-19990114>. Acesso em: 06 jun. 2003.

BRAY, Tim (Ed.) et al. **Extensible Markup Language (XML) 1.0**. 2 ed., 2000. Disponível em: <http://www.w3.org/TR/REC-xml>. Acesso em: 05 jun. 2003.

BRITISH STANDARDS. **What are Standards**. 2003. Disponível em: <http://www.bsi-global.com/All+About+Standards/What+are+Standards/What.xalter>. Acesso em: 02 out. 2003.

CABRAL, Rodrigo B.; SPERB, Rafael M.; TRIPODI, Rodrigo Z. **Tecnologia Opensource para Rastreamento Online de Veículos Monitorados via Satélite**. Itajaí. 2002. 11p. Disponível em: <http://g10.cttmar.univali.br/publicacao-01.pdf>. Acesso em: 20 mai. 2003.

CABRAL, Rodrigo B.; et al. **RASTRO: Internet Based Tracking System for Fisheries Control.** In: FIFTH INTERNATIONAL SYMPOSIUM ON GIS AND COMPUTER CARTOGRAPHY FOR COASTAL ZONE MANAGEMENT, October 2003, Genova, Italy. CD-ROM Proceedings of the Fifth International Symposium on GIS and Computer Cartography for Coastal Zone Management. 2003.

CERAMI, Ethan. **Web Services Essentials**. CA. O'Reilly, 2002. ISBN 0-596-00224-6.

CERÍCOLA, V. O. **Oracle: Banco de Dados Relacional e Distribuído**. São Paulo: Makron Books, 1995. 448 p.

CUNHA, Davi. **Web Services, SOAP e Aplicações Web**. Disponível em: <http://devedge.netscape.com/viewsource/2002/soap-overview/index_pt_br.html>. Acesso em: 12 jun. 2003.

DAVE, Vasant. **The Third Eye of the Water Supply Manager - Radio Telemetry**. Disponível em: <http://home.123india.com/vasantdave/EYENOPIC.htm>. Acesso em: 01 mai. 2003.

DUMBILL, Edd. **Backends Sharing Data**. 1999. Disponível em: <http://www.xml.com/pub/a/1999/08/rpc/index.html>. Acesso em: 06 jun. 2003.

ERICSSON. **Customer Perspective**. Disponível em: <http://www.ericsson.com/network_operators/mobitex/subpages/mnews/cp_01.shtml>. Acesso em: 01 mai. 2003.

FURLAN, José Davi. **Modelagem de Objetos através da UML: Análise e Projeto Orientados a Objeto**. São Paulo: Makron Books. 1998. 329 p.

GREGO, Maurício. De carona com Web Services. **Revista INFO Corporate**, São Paulo: Abril, n.1, p. 28-37, dez. 2002.

IEC - International Electrotechnical Commission. **Mission and Objectives**. 2003. Disponível em: <http://www.iec.ch/about/mission-e.htm>. Acesso em: 04 out. 2003.

IN4MA REMOTE MONITORING SOLUTIONS. **What is Telemetry**. Disponível em: <http://www.in4ma.co.uk/faq/telemetry.html>. Acesso em: 15 abr. 2003.

ISO - International Organization for Standardization. **Introduction**. 2003a. Disponível em: <http://www.iso.ch/iso/en/aboutiso/introduction/index.html>. Acesso em: 03 out. 2003.

ISO - International Organization for Standardization. **Stage of the development of International Standards**. 2003b. Disponível em: < http://www.iso.ch/iso/en/stdsdevelopment/whowhenhow/proc/proc.html>. Acesso em: 14 dez. 2003.

JACOBS, Ian. **About the World Wide Web Consortium (W3C)**. 2000. Disponível em: <http://www.w3.org/Consortium/>. Acesso em: 04 out. 2003.

LATTEIER, Amos. **Internet Scripting: Zope and XML-RPC**. Disponível em: <http://www.xml.com/pub/a/2000/01/xmlrpc/> Acesso em: 06 jun. 2003.

L-3 COMMUNICATIONS TELEMETRY & INSTRUMENTATION. **Telemetry Tutorial**. Disponível em: <http://www.ti.l-3com.com/tutorial/preface.html>. Acesso em: 09 mar. 2003.

KIDD, Eric. **XML-RPC Howto**. 2001. Disponível em: <http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html>. Acesso em: 06 jun. 2003.

MYSQL AB, **Manual de Referência do MySQL**. 2003. Disponível em: <http://www.mysql.com/doc/pt>. Acesso em: 11 nov. 2003.

NASA AMES RESEARCH CENTER. **Principles of Operations**. Disponível em: <http://ic.arc.nasa.gov/ic/projects/remote-agent/activities/pofo/docs/Communications/1-what-is-telemetry.html>. Acesso em: 08 mar. 2003.

NERY, Fernando; PARANHOS, Maurício. **Cobit ou ISO 17799? Iniciando a Reflexão**. 2003. 10p. Disponível em : <http://www.modulo.com.br/pdf/cobit_iso.pdf>. Acesso em: 04 out. 2003.

NMEA - National Marine Electronics Association. **NMEA 2000 Standard**. Disponível em: <http://www.nmea.org/>. Acesso em: 11 mar. 2003.

OMNITOR AB. **What is standardization and why you have to follow technical standards**. 2003. Disponível em: <http://www.omnitor.se/english/standards/>. Acesso em: 02 out. 2003.

OPEN GIS Consortium. **The OGC Technical Committee Policies & Procedures**. 2000. disponível em: <http://www.opengis.org/docs/pnp.pdf>. Acesso em: 14 dez. 2003.

OPEN GIS Consortium. **Specifications Program**. 2003. disponível em: < http://www.opengis.org/about/?page=spec>. Acesso em: 14 dez. 2003.

OPV - Oficina Projeto Virtual. Web Services. Disponível em: <http://www.iweb.com.br/opv/tec_web.php3>. Acesso em: 05 mai. 2003.

PHP Group. **What is PHP**. 2003a. Disponível em: <http://www.php.net>. Acesso em: 08 nov. 2003.

PHP Group. **About PEAR**. 2003b. Disponível em: <http://pear.php.net>. Acesso em: 08 nov. 2003.

RHODES, Kate. **XML-RPC vs. SOAP**. Disponível em: <http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm>. Acesso em: 13 jun. 2003.

SEBRAE-SC - Serviço de Apoio às Micros e Pequenas Empresas em Santa Catarina. **O que é Normalização?**. 2003. Disponível em: <http://www.sebrae-sc.com.br/sebraetib/conceitos/normalizacao/meiocertif.html>. Acesso em: 03 out. 2003.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 3. ed. São Paulo: Makron Books, 1999. 778 p

SRC - SEISMOLOGY RESEARCH CENTRE. **Technical Notes: Internet Telemetry**. Disponível em: <http://www.seis.com.au/TechNotes/TN200302B_INet_Telem.html>. Acesso em: 01 mai. 2003.

TESCH, José R. **XML Schema**. Florianópolis: Ed. Visual Books, 2002. ISBN 85-7502-073-0.

THOMPSON, Henry S. et al. **XML Schema Part 1: Structures**. 2001. Disponível em: <http://www.w3.org/TR/xmlschema-1/>. Acesso em: 06 jun. 2003.

TIDWELL, Doug. **Programming Web Services with SOAP**. Ed. O'Reilly. 2001. ISBN 0-596-00095-2.

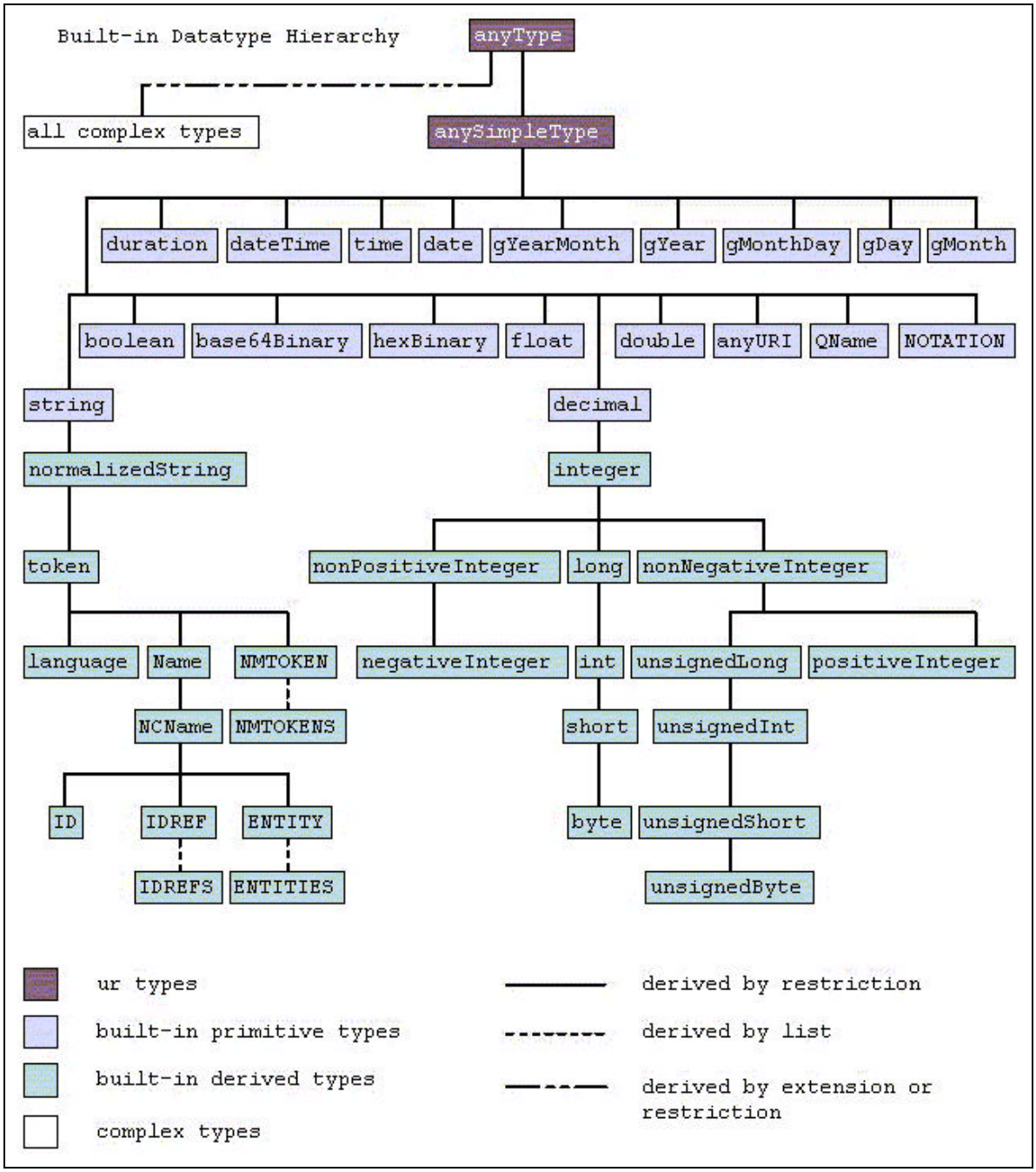
USERLAND Software. **XML-RPC Home Page**. 2001. Disponível em: <http://www.xmlrpc.com/#whatIsXmlrpc>. Acesso em: 10 mai. 2003.

WINER, Dave. **XML-RPC Specification**. 1999. Disponível em: <http://www.xmlrpc.com/spec>. Acesso em: 06 jun. 2003.

# ANEXOS

**ANEXO I - Árvore hierárquica dos tipos de dados embutidos no XML *Schema*.**

O Diagrama a seguir apresenta a árvore hierárquica dos tipos de dados embutidos no XML *Schema* que foi citado no item 3.3.2.1 - Tipos de Dados Simples e Complexos (BIRON, 2001).



**ANEXO II - Exemplo da estrutura de um documento WSDL.**

O código apresentado a seguir refere-se ao exemplo utilizado para apresentar a estrutura de um documento WSDL no item 3.5.2 - Estrutura de um documento WSDL (AYALA, 2002, p.77).

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:tns="http://abcom.com/stocktrading.wsdl"
xmlns:xsd1="http://abcom.com/stocktrading.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://abcom.com/stocktrading.wsdl" name="StockTrading">
   <types>
      <schema targetNamespace="http://abcom.com/stocktrading.xsd"
       xmlns="http://www.w3.org/2000/10/XMLSchema">
         <element name="StockRateRequest">
            <complexType>
               <all>
                  <element name="StockScript" type="string"/>
               </all>
            </complexType>
         </element>
         <element name="StockRate">
            <complexType>
               <all>
                  <element name="rate" type="float"/>
               </all>
            </complexType>
         </element>
      </schema>
   </types>
   <message name="GetStockRateInput">
      <part name="body" element="xsd1:StockRateRequest"/>
   </message>
   <message name="GetStockRateOutput">
      <part name="body" element="xsd1:StockRate"/>
   </message>
   <portType name="StockTradingPortType">
      <operation name="GetStockRate">
         <input message="tns:GetStockRateInput"/>
         <output message="tns:GetStockRateInput"/>
      </operation>
   </portType>
   <binding name="StockTradingSoapBinding" type="tns:StockTradingPortType">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetStockRate">
         <soap:operation soapAction="http://abcom.com/GetStockRate"/>
         <input>
            <soap:body use="literal"/>
         </input>
         <output>
            <soap:body use="literal"/>
         </output>
      </operation>
   </binding>
   <service name="StockTradingService">
      <documentation>FIRST SERVICE</documentation>
      <port name="StockTradingPort" binding="tns:StockTradingSoapBinding">
         <soap:address location="http://abcom.com/stocktrading"/>
      </port>
   </service>
</definitions>
```

**ANEXO III - Documento WSDL que descreve o *Web Service* do modelo proposto.**

O código a seguir, representa o documento WSDL que descreve o modelo de *Web Service* desenvolvido por este trabalho.

```
<?xml version="1.0"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:WS_telemetria"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns5="http://localhost/rastro/xsd" targetNamespace="urn:WS_telemetria"
name="WS_telemetria">
   <wsdl:types>
      <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://localhost/rastro/xsd">
         <complexType name="dad_sensor">
            <all>
               <element name="lei_med" type="xsd:float"/>
               <element name="sen_cod" type="xsd:int"/>
            </all>
         </complexType>
         <complexType name="array_sensor">
            <complexContent>
               <restriction base="SOAP-ENC:Array">
                  <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="ns5:dad_sensor[]"/>
               </restriction>
            </complexContent>
         </complexType>
         <complexType name="dados_ocorrencia">
            <all>
               <element name="col_cod" type="xsd:int"/>
               <element name="reg_lat" type="xsd:float"/>
               <element name="reg_lon" type="xsd:float"/>
               <element name="oco_dat" type="xsd:datetime"/>
               <element name="arr_sen" type="ns5:array_sensor"/>
            </all>
         </complexType>
         <complexType name="campos_coleta">
            <all>
               <element name="col_cod" type="xsd:int"/>
               <element name="col_nom" type="xsd:string"/>
               <element name="col_des" type="xsd:string"/>
            </all>
         </complexType>
         <complexType name="registro_coleta">
            <complexContent>
               <restriction base="SOAP-ENC:Array">
                  <attribute ref="SOAP-ENC:arrayType"
wsdl:arrayType="ns5:campos_coleta[]"/>
               </restriction>
            </complexContent>
         </complexType>
         <complexType name="campos_registro">
            <all>
               <element name="reg_cod" type="xsd:int"/>
               <element name="reg_dat" type="xsd:string"/>
               <element name="reg_lat" type="xsd:float"/>
               <element name="reg_lon" type="xsd:float"/>
               <element name="lei_med" type="xsd:float"/>
               <element name="sen_cod" type="xsd:int"/>
            </all>
         </complexType>
         <complexType name="reg_registro">
            <complexContent>
               <restriction base="SOAP-ENC:Array">
                  <attribute ref="SOAP-ENC:arrayType"
wsdl:arrayType="ns5:campos_registro[]"/>
               </restriction>
            </complexContent>
```

```
         </complexType>
         <complexType name="campos_sensor">
            <all>
               <element name="sen_cod" type="xsd:int"/>
               <element name="sen_nom" type="xsd:string"/>
               <element name="sen_des" type="xsd:string"/>
               <element name="sen_uni" type="xsd:string"/>
            </all>
         </complexType>
         <complexType name="registro_sensor">
            <complexContent>
               <restriction base="SOAP-ENC:Array">
                  <attribute ref="SOAP-ENC:arrayType"
wsdl:arrayType="ns5:campos_sensor[]"/>
               </restriction>
            </complexContent>
         </complexType>
      </schema>
   </wsdl:types>
   <message name="set_ocorrenciaRequest">
      <part name="inputStruct" type="ns5:dados_ocorrencia"/>
   </message>
   <message name="set_ocorrenciaResponse">
      <part name="op_cod" type="xsd:string"/>
   </message>
   <message name="get_coleta_infRequest">
      <part name="cli_cod" type="xsd:int"/>
   </message>
   <message name="get_coleta_infResponse">
      <part name="outputStruct" type="ns5:registro_coleta"/>
   </message>
   <message name="get_registro_infRequest">
      <part name="col_cod" type="xsd:int"/>
   </message>
   <message name="get_registro_infResponse">
      <part name="outputStruct" type="ns5:reg_registro"/>
   </message>
   <message name="get_sensor_infRequest">
      <part name="col_cod" type="xsd:int"/>
   </message>
   <message name="get_sensor_infResponse">
      <part name="outputStruct" type="ns5:registro_sensor"/>
   </message>
   <portType name="WS_telemetriaPort">
      <operation name="set_ocorrencia">
         <input message="tns:set_ocorrenciaRequest"/>
         <output message="tns:set_ocorrenciaResponse"/>
      </operation>
      <operation name="get_coleta_inf">
         <input message="tns:get_coleta_infRequest"/>
         <output message="tns:get_coleta_infResponse"/>
      </operation>
      <operation name="get_registro_inf">
         <input message="tns:get_registro_infRequest"/>
         <output message="tns:get_registro_infResponse"/>
      </operation>
      <operation name="get_sensor_inf">
         <input message="tns:get_sensor_infRequest"/>
         <output message="tns:get_sensor_infResponse"/>
      </operation>
   </portType>
   <binding name="WS_telemetriaBinding" type="tns:WS_telemetriaPort">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="set_ocorrencia">
         <soap:operation
soapAction="http://localhost/rastro/servidor#metodos#set_ocorrencia "/>
         <input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/ "
namespace="http://localhost/rastro/servidor"/>
         </input>
         <output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/ "
namespace="http://localhost/rastro/servidor"/>
         </output>
```

```
      </operation>
      <operation name="get_coleta_inf">
         <soap:operation
soapAction="http://localhost/rastro/servidor#metodos#get_coleta_inf"/>
         <input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost/rastro/servidor"/>
         </input>
         <output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost/rastro/servidor"/>
         </output>
      </operation>
      <operation name="get_registro_inf">
         <soap:operation
soapAction="http://localhost/rastro/servidor#metodos#get_registro_inf"/>
         <input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost/rastro/servidor"/>
         </input>
         <output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost/rastro/servidor"/>
         </output>
      </operation>
      <operation name="get_sensor_inf">
         <soap:operation
soapAction="http://localhost/rastro/servidor#metodos#get_sensor_inf"/>
         <input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost/rastro/servidor"/>
         </input>
         <output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost/rastro/servidor"/>
         </output>
      </operation>
   </binding>
   <service name="WS_telemetriaService">
      <documentation/>
      <port name="WS_telemetriaPort" binding="tns:WS_telemetriaBinding">
         <soap:address location="http://localhost/rastro/wsserver.php"/>
      </port>
   </service>
</wsdl:definitions>
```

# ANEXO IV – Padrão Proposto: *Web Service* para Transferência de Dados Telemétricos

# AUTORIZAÇÃO

**PROPOSTA DE UM PADRÃO PARA A COMUNICAÇÃO**
**DE DADOS EM SISTEMAS DE TELEMETRIA BASEADO EM WEB SERVICES**

Pela presente autorizamos a UNIVALI - UNIVERSIDADE DO VALE DO ITAJAÍ a expor em suas dependências o presente relatório do Estágio Supervisionado efetuado pelo acadêmico José Morelli Neto, durante o período de _____ à _____.

_____

Anita Maria da Rocha Fernandes, Dra
Coordenadora de Estágio Supervisionado

# TERMO DE APROVAÇÃO

**PROPOSTA DE UMA PADRÃO PARA A COMUNICAÇÃO DE DADOS EM SISTEMAS DE TELEMETRIA BASEADO EM WEB SERVICES**

José Morelli Neto

Este Relatório foi julgado adequado para obtenção dos créditos da disciplina de Estágio Supervisionado do curso de Ciência da Computação.

_____

Rodrigo Becke Cabral, Dr.
Orientador

_____

Profª. Anita Maria da Rocha Fernandes, Dra.
Coordenadora de Estágio

BANCA EXAMINADORA

_____

Cesar Albenes Zeferino, Dr.

_____

Rafael Luiz Cancian, M.Sc.

Itajaí, dezembro de 2003.

**ANEXO V – Artigo – Proposta de um Padrão para a Comunicação de Dados em Sistemas de Telemetria baseado em Web Services**