Guilherme Bertoni Machado

Uma Arquitetura baseada em Web Services com Diferenciação de Serviços para Integração de Sistemas Embutidos a outros Sistemas

Florianópolis - SC

2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Guilherme Bertoni Machado

Uma Arquitetura baseada em Web Services com Diferenciação de Serviços para Integração de Sistemas Embutidos a outros Sistemas

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Frank Augusto Siqueira

Florianópolis, junho de 2006

Uma Arquitetura baseada em Web Services com Diferenciação de Serviços para Integração de Sistemas Embutidos a outros Sistemas

Guilherme Bertoni Machado

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

____

Prof. Dr. Raul Sidnei Wazlawick

Coordenador do Curso

Banca Examinadora

____

Prof. Dr. Frank Augusto Siqueira (Orientador)

____

Prof. Dr. Antônio Augusto Medeiros Fröhlich

____

Prof. Dr. Carlos Barros Montez

____

Prof. Dr. João Bosco Mangueira Sobral

____

Prof. Dr. Lau Cheuk Lung

Agradecimentos

Como este é um espaço menos formal, irei agradecer aqueles que de uma forma ou outra me ajudaram nessa pelea, se eu acabar esquecendo de alguém foi sem querer.

Primeiramente tenho, e muito, que agradecer ao Frank. Baita orientador, super dedicado, sempre presente, e além de tudo gente finíssima. Aproveito o momento para agradecer os professores da banca, Guto, Bosco Sobral, Montez e Lau, que, com certeza, através das suas contribuições ajudaram a melhorar esta dissertação. Também agradeço aos professores Bosco Alves, Mazzola e Dantas, pela atenção dedicada.

Não posso deixar de elogiar a Universidade Federal de Santa Catarina, por me oferecer, durante 7 anos (graduação, especialização e mestrado) toda a estrutura necessária para o meu crescimento pessoal e profissional.

Ao pessoal do departamento (valeu Verinha!) e do LaPeSD, principalmente a Lea e a Fabri, minhas colegas de primeiro ano, meu muito obrigado.

Sou muito grato aos meus amigos, principalmente por não perguntarem sobre a minha dissertação, toda a gurizada de Pelotas - Digão, Ney, Peter, Bó, Rick, Chancho, Fabrício, Gleifer, e tantos outros, bem como a minha turma de Floripa - Zé, Petrucio, Pedro, Arliones, Cabeca, Frango, Pretinho, Ivanrs, Capin, Carlucci, Rato, Salsa, e por ai vai. Vocês são a família que eu escolhi.

Por falar em família, não tenho palavras para demonstrar a gratidão pela força, em todos os sentidos, que vocês me deram. Mãe, João, Pai, Sandra, Fi, Guga, Cris, Lulu, Johnny, Beta, tias Leni, Lenice e Iria um grande beijo e um abraço bem forte.

É isso, e que venham novos desafios!

Sumário

Lista de Figuras

Lista de Abreviaturas

Resumo

Abstract

1 Introdução p. 15

1.1 Motivação . . . . p. 17

1.2 Objetivos . . . . p. 18

1.2.1 Objetivo Geral . . . . p. 18

1.2.2 Objetivos Específicos . . . . p. 19

1.3 Justificativa . . . . p. 19

1.4 Metodologia . . . . p. 19

1.5 Resultados Esperados . . . . p. 20

1.6 Estrutura do Documento . . . . p. 20

2 Revisão da Literatura p. 21

2.1 *Web Services* . . . . p. 21

2.1.1 Principais Tecnologias Empregadas por *Web Services* . . . . p. 22

2.1.2 XML - *Extensible Markup Language* . . . . p. 24

2.1.3 SOAP - *XML Protocol* . . . . p. 25

2.1.4 WSDL - *Web Service Description Language* . . . . p. 27

2.1.5 UDDI - *Universal Description, Discovery, and Integration* . . . . . . p. 29

2.2 Qualidade de Serviço em *Web Services* . . . . p. 30

2.2.1 Especificações de QoS . . . . p. 31

2.3 Conceitos de Sistemas de Tempo Real . . . . p. 32

2.4 Sistemas Embutidos . . . . p. 34

2.4.1 Características dos Sistemas Operacionais para Sistemas Embutidos . p. 36

2.4.2 Sistemas Embutidos e a Internet . . . . p. 36

2.5 Considerações Finais . . . . p. 37

3 Implantando *Web Services* em Sistemas Embutidos p. 38

3.1 Plataformas de conectividade . . . . p. 39

3.2 *Toolkits* de Desenvolvimento . . . . p. 41

3.3 SHIP . . . . p. 42

3.4 gSOAP . . . . p. 43

3.5 Porte, adaptação do *Toolkit* para a Plataforma . . . . p. 46

3.6 Considerações Finais . . . . p. 48

4 Diferenciação de Serviços em Web Services p. 49

4.1 Solução . . . . p. 50

4.2 Características da Abordagem Proposta . . . . p. 53

4.3 Considerações Finais . . . . p. 55

5 Implantação e Análise dos Resultados p. 56

5.1 Casos de Uso . . . . p. 58

5.1.1 Integração de um Sistema de Controle de Redes Industriais . . . . . . p. 58

5.1.1.1 Serviço Implementado . . . . p. 60

5.1.2 Utilização de Sistemas Embutidos no Monitoramento e Controle de Pacientes . . . . p. 62

5.1.2.1 Serviço Implementado . . . . p. 63

5.2 Avaliação do Ambiente de Execução . . . . p. 64

5.2.1 JMeter . . . . p. 64

5.2.2 Testes de Desempenho . . . . p. 65

5.2.3 Testes do Classificador . . . . p. 67

5.2.3.1 Impacto do classificador no desempenho . . . . p. 68

5.3 Considerações Finais . . . . p. 68

6 Conclusões p. 70

6.1 Limitações do Trabalho . . . . p. 72

6.2 Perspectivas Futuras . . . . p. 73

Referências p. 74

Anexo I - Configuração, Instalação e Utilização do gSOAP p. 80

Anexo II - Exemplo de Aplicação p. 83

Anexo III - Arquivos de configuração do gSOAP modificados p. 86

Anexo IV - Makefiles p. 92

Lista de Figuras

2.1 Arquitetura SOA . . . . p. 21

2.2 Exemplo de interação entre as tecnologias de um *Web Service*. . . . . p. 23

2.3 Camadas da arquitetura de *Web Service*. . . . . p. 23

2.4 Documento XML. . . . . p. 24

2.5 Estrutura de uma mensagem SOAP. . . . . p. 26

2.6 Estrutura de um documento WSDL. . . . . p. 27

2.7 WSDL . . . . p. 28

2.8 Descoberta de *Web Services* através de um repositório UDDI. . . . . p. 29

2.9 Sistema de Tempo Real. . . . . p. 33

2.10 Sistema Embutido . . . . p. 35

3.1 Estrutura de um *Web Service* embutido . . . . p. 39

3.2 SHIP - Software e Hardware Integrados em uma Plataforma . . . . p. 42

3.3 Desenvolvendo e Disponibilizando um serviço com gSOAP . . . . p. 44

3.4 Desenvolvendo e Disponibilizando um cliente com gSOAP . . . . p. 44

4.1 Arquitetura do Ambiente . . . . p. 49

4.2 Abordagem de escalonamento . . . . p. 51

4.3 Sistema proposto. . . . . p. 54

5.1 Criação do Ambiente de Execução - com ou sem classificação do métodos . . p. 57

5.2 Compilação do Web Service . . . . p. 58

5.3 Cenário de Integração em Sistemas de Controle . . . . p. 59

5.4 Definições em C . . . . p. 60

5.5 WSDL gerado a partir da definição dos serviços e tipos dos dados . . . . . . p. 61

5.6 Cenário de Utilização para Monitoramento e Controle de Pacientes . . . . . . p. 62

5.7 Definições em C . . . . p. 63

5.8 Configuração do JMeter . . . . p. 64

5.9 Tempo de Resposta - 1, 2 e 4 clientes sem concorrência . . . . p. 65

5.10 Vazão e Número de Ocorrências - 1, 2 e 4 clientes sem concorrência . . . . . p. 66

5.11 4 clientes de classes diferentes e com concorrência . . . . p. 67

Lista de Abreviaturas

ARM : Advanced RISC Machines

API : Application Programming Interface

CAN : Controlled Area Networks

COM : Component Object Model

CORBA : Common Object Request Broker Architecture

CPU : Central Processing Unit

DCOM : Distributed Component Object Model

DVD : Digital Video Disc

EPROM : Erasable Programmable Read-only Memory

ETTK : Emerging Technologies Toolkit

FTP : File Transfer Protocol

HTML : Hyper-Text Markup Language

HTTP : Hyper-Text Transfer Protocol

IBM : International Business Machines

IDL : Interface Description Language

IIS : Internet Information Services

J2ME : Java Platform, Micro Edition

LCD : Light-emitting Diode

LED : Liquid Crystal Display

MSW : MicroServidor Web

PDA : Personal Digital Assistants

PROFIBUS : Process Field Bus

QoS : Quality of Service

RAM : Random-access Memory

RISC : Reduced Instruction Set Computer

RMI : Remote Method Invocation

ROM : Read-only Memory

RPC : Remote Procedure Call

RTOS : Real-time Operating System

SHIP : Software e Hardware Integrados em uma Plataforma

SMTP : Simple Mail Transfer Protocol

SaaS : Software as a Service

SO : Sistema Operacional

SOA : Service-Oriented Architecture

SOAP : XML Protocol / Simple Object Access Protocol (até versão 1.1)

STR : Sistema de Tempo Real

TCP/IP : Transmission Control Protocol / Internet Protocol

UDDI	: Universal Description Discovery Integration
uPnP	: Universal Plug and Play
W3C	: World Wide Web Consortium
WSDL	: Web Services Description Language
XML	: Extensible Markup Language

Resumo

Tecnologias para integração de sistemas, como *Web Services*, vêm sendo empregadas com sucesso para integração de softwares empresariais, permitindo a interação entre sistemas utilizados em diferentes organizações. *Web Services* têm se mostrado uma arquitetura eficiente para a interconexão de sistemas através da rede.

Sistemas Embutidos estão cada vez mais integrados à Internet através da interconexão destes dispositivos em redes TCP/IP. E a integração de aplicações provenientes dos Sistemas Embutidos a outros sistemas vem se mostrando, do mesmo modo, cada vez mais necessária.

Este trabalho busca propor e apresentar uma arquitetura baseada em *Web Services* com diferenciação de serviços para integração de Sistemas Embutidos a outros sistemas. Portanto, realizamos uma definição de uma política de escalonamento com diferenciação entre os serviços e analisamos a adequação da arquitetura de *Web Services* e da infra-estrutura desenvolvida neste trabalho para a integração de aplicações desenvolvidas sobre sistemas embutidos.

A implantação desta arquitetura foi alcançada através do estudo, da modelagem e desenvolvimento do suporte para execução de *web services* projetados com o *toolkit* gSOAP tendo como ambiente o sistema embutido SHIP, identificando as limitações para sua integração com outros dispositivos.

Ao longo deste trabalho, realizamos mudanças no *firmware* deste dispositivo com o intuito de permitir a sua integração através do uso de *Web Services*, e também criamos uma extensão ao *toolkit* gSOAP para que este oferecesse suporte a diferenciação de serviços.

Através da implementação dos serviços usados como testes, conseguimos demonstrar que podemos disponibilizar uma plataforma para o desenvolvimento de *Web Services* em sistemas embutidos, tanto em relação ao desempenho, quanto ao classificador de serviços proposto, sendo viável para um conjunto significativo de aplicações com restrições temporais do tipo melhor-esforço e de tempo real brando (*soft*).

Palavras-chave: *Web Services*, Sistemas Embutidos, gSOAP, QoS, Tempo Real

Abstract

System integration technologies, such as Web Services, have been employed successfully for the integration of business software, allowing the interaction between systems hosted by different companies. Web Services has shown an efficient architecture for the interconnection of systems through the net.

Embedded Systems are more integrated to the Internet through the interconnection of these devices in TCP/IP nets. And the integration of applications proceeding from the Embedded Systems to other systems, in a similar way, becomes more necessary.

This work seeks to propose and to present an architecture based on Web Services with services differentiation for Embedded Systems integration to other systems. Therefore, we made a scheduling politics definition with differentiation between the services and analyze the adequacy of the Web Services architecture and the infrastructure developed in this work for the integration of applications developed on Embedded systems.

The implantation of this architecture was reached through the study, modeling and development of a web services execution support projected with the gSOAP toolkit having as embedded system environment the SHIP board, identifying the limitations for its integration with other devices.

Along these work, we made changes in the firmware of these device in order to allow their integration through the use of Web Services, and also we create an gSOAP's toolkit extension to provide services differentiation support.

Through the services implementation used as tests, we demonstrate that we can provide a platform for the development of Web Services in Embedded Systems, achieving performance and classifying services, being viable for a significant set of applications with time constraints such as best-effort and soft real-time.

Keywords: Web Services, Embedded Systems, gSOAP, QoS, Real-time

1 Introdução

A *World Wide Web* teve como objetivo inicial permitir a troca de documentos entre os computadores distribuídos por essa rede. Porém, com o crescimento e popularização desta, passou-se a utilizá-la como base para comunicação entre aplicações distribuídas que necessitam de um método eficiente para intercâmbio de dados (informações).

Conseqüentemente, uma parcela significativa dos sistemas computacionais passou a utilizar este modelo de comunicação, permitindo vislumbrar a integração destes, uma vez que uma das maiores dificuldades existentes atualmente no mundo computacional é a integração de sistemas.

Tecnologias para integração buscam prover, em diferentes níveis, meios de realizar trocas de informação entre aplicações distribuídas em um ambiente heterogêneo. Dentre as tecnologias existentes destacamos a *Universal Plug and Play device architecture* - uPnP (UPnP Forum, 2000), utilizada para descoberta e auto-configuração de dispositivos; *Common Object Request Broker Architecture* (CORBA) (OMG, 2005), utilizada para integração de sistemas computacionais orientadas a objetos; e *Web Services* (BOOTH et al., 2004), principal objeto de estudo deste trabalho.

A Arquitetura de *Web Services*[1] busca a solução sobre o problema da integração, pois ao utilizar padrões abertos de protocolos e linguagens, possibilita a integração das mais diversas aplicações distribuídas sem se preocupar com a heterogeneidade intrínseca dos ambientes distribuídos (como a *Web*, por exemplo) (BONIATI; PADOIN, 2003).

[1] Na literatura pesquisada também foram encontrados os termos *WebServices* e Serviços Web (termo em português). Optou-se por manter a grafia proposta pelo W3C (W3C, 2005d), e *web service* como uma instância (aplicação) de um *Web Service*.

Web Services apresentam-se como uma evolução das tecnologias de comunicação baseadas em Objetos Distribuídos (VOGELS, 2003), como por exemplo CORBA (OMG, 2005), Java RMI (SUN MICROSYSTEMS, 2003) e COM/DCOM (Microsoft, 1996). No entanto, ao invés de fazer referência a uma interface de um objeto, um *web service* busca uma mudança deste paradigma para uma Arquitetura Orientada a Serviços (SOA - *Service-Oriented Architecture*) (CHAPPELL; JEWELL, 2002).

Devido à adoção da arquitetura SOA, os sistemas (*softwares*) deixam de ser orientados a objetos[2] para serem compostos por vários serviços descritos por um ou mais *web services* (LUCCA, 2003).

Além disso, *Web Services* são um meio capaz de prover a interação de aplicações e programas sem a intermediação do homem (LEA; VINOSKI, 2003), isto é, *web services* podem se comunicar com outros *web services*, podendo assim fornecer novas aplicações com os serviços provenientes destes.

Um exemplo dessa prática seria um portal contendo um *web service* que pesquisa diversas empresas aéreas com o intuito de organizar programas especiais de final de semana. Se previamente o usuário informar que gostaria de alugar um carro na cidade escolhida, fazer a reserva de uma mesa em um restaurante, realizar a compra de ingressos para o teatro e realizar a pesquisa de hotéis dentro da faixa de preços informada, o *web service* da empresa aérea escolhida submete (envia) as informações (dados) aos outros *web services* responsáveis pelas tarefas acima citadas e efetua as requisições do cliente.

Por outro lado, a maioria dos equipamentos que fazem parte do nosso dia-a-dia - telefones celulares, eletroeletrônicos, aparelhos de áudio e vídeo, PDAs (todos estes sistemas embutidos) e, é claro, computadores pessoais - não estão totalmente integrados, necessitando que estes sejam manuseados individualmente através de suas próprias interfaces. A comunicação entre estes dispositivos é possível devido à disponibilidade de conexões de rede entre a maioria destes

[2]Nos referimos com relação à construção dos sistemas, e não à programação e nem ao desenvolvimento de sistemas orientados a objetos, mas sim em relação à forma como são modelados os componentes de um sistema computacional distribuído (numa aplicação CORBA, por exemplo) (LUCCA, 2003).

dispositivos através de redes com ou sem fio, mas isto não é tudo. Cientistas vêm pesquisando meios de permitir que estes dispositivos interajam - e não somente se comuniquem - através de interfaces e protocolos de comunicação padronizados a fim de que estes possam trabalhar conjuntamente em um ambiente integrado. Isto irá, em um futuro próximo, permitir que usuários controlem facilmente estes equipamentos através de qualquer outro que esteja conectado na mesma rede. Esta revolução foi chamada de "computação ubíqua" por Mark Weiser no começo dos anos noventa (WEISER, 1993) e, hoje em dia, também é nomeada de "computação pervasiva".

Sabendo de toda a potencialidade que *Web Services* podem oferecer e analisando a tendência de uma maior integração entre Software e Hardware, nada mais natural em se propor a construção/implementação de *web services* também em sistemas embutidos, uma vez que estes representam a maior fatia do mercado de processadores e suas aplicações estão cada vez mais sofisticadas, principalmente pelo fato de que estes estão cada vez mais integrados à Internet através da sua interconexão em redes TCP/IP.

Como conseqüência deste processo de integração torna-se possível para as aplicações provenientes dos Sistemas Embutidos interagirem com aplicações disponíveis em outros sistemas, se aproximando cada vez mais do conceito de computação pervasiva, aumentando mais ainda a gama de dispositivos que podem compor um ambiente inteligente, conforme apresentado por Issarny et al. (2002).

1.1 Motivação

Prover a integração dos sistemas e plataformas computacionais atualmente disponíveis é um dos objetivos mais pesquisados pelos cientistas da computação, e que começa a se tornar uma realidade hoje em dia. Tecnologias para integração de sistemas, como *Web Services*, podem prover um *middleware* para sistemas que originalmente eram independentes.

Esta tecnologia tem sido empregada com sucesso para integração de softwares empresariais (VINOSKI, 2003), permitindo a interação entre sistemas utilizados em diferentes companhias.

Apesar do alto nível de integração alcançado nas interações entre empresas, a mesma integração nem sempre é obtida nos diferentes níveis internos de uma corporação. Neste ambiente, diversos dispositivos independentes (com grande destaque para dispositivos baseados em sistemas embutidos) e sistemas de comunicação coexistem, e sua integração geralmente requer soluções customizadas.

Sistemas Embutidos estão cada vez mais integrados à Internet através da interconexão destes dispositivos em redes TCP/IP. *Web Services* têm se mostrado uma arquitetura eficiente para a interconexão de sistemas através da rede; por outro lado, a integração de aplicações provenientes dos Sistemas Embutidos a outros sistemas vem se mostrando, do mesmo modo, cada vez mais necessária.

A partir desse trabalho, esperamos possibilitar uma nova área de aplicação dos *Web Services* como meio de integração de Sistemas Embutidos a outros sistemas, podendo assim promover a integração destes em um ambiente distribuído.

Além disso, este trabalho se propõe a adicionar suporte a qualidade de serviço em *Web Services*, principalmente no que diz a respeito à classificação destes serviços, fundamental para o aspecto temporal, visto que requisitos de QoS e sua política de utilização ainda não estão bem consolidados no ambiente *Web* e que geralmente requisitos de tempo são importantíssimos em Sistemas Embutidos.

1.2 Objetivos

Em seguida serão apresentados os objetivos deste trabalho, classificando-os entre objetivo geral e específicos.

1.2.1 Objetivo Geral

Este trabalho visa propor e apresentar uma infra-estrutura para suporte ao desenvolvimento de *Web Services* em uma plataforma embutida, com o intuito de integrá-los a outros sistemas.

1.2.2 Objetivos Específicos

Consistem em objetivos específicos do trabalho:

1. Definição de uma política de escalonamento com diferenciação entre os serviços e/ou usuários destes, de modo a disponibilizar um suporte para os requisitos temporais impostos por aplicações para as quais o fator tempo é essencial;

2. Analisar a adequação da arquitetura de *Web Services* e da infra-estrutura desenvolvida neste trabalho para a integração de aplicações desenvolvidas sobre sistemas embutidos.

1.3 Justificativa

É cada vez maior o interesse acadêmico e do mercado pela utilização de *Web Services*, assim como é grande o interesse pela integração dos Sistemas Embutidos ao mundo exterior, já integrado pela Internet.

Por isso, buscamos nesse trabalho, agregar essas duas visões, ou seja, lançar mão dos *Web Services* para a integração de Sistemas Embutidos a outros sistemas, assunto bastante recente, e de extrema relevância.

Além disso, devido à necessidade de garantir o provimento de requisitos temporais existentes em grande parte dos sistemas embutidos, buscamos integrar suporte a QoS em *Web Services*, o que torna o trabalho bastante relevante visto que há pouca pesquisa em relação a essa abordagem.

1.4 Metodologia

Numa primeira etapa será realizado o estudo das tecnologias envolvidas e serão identificadas suas limitações para disponibilização de *Web Services* em sistemas embutidos.

Em seguida, realizaremos a definição de todo um suporte para o desenvolvimento de *Web*

Services em uma plataforma embutida, definindo qual o *toolkit* de desenvolvimento a ser utilizado e identificando as adequações necessárias para sua utilização em sistemas embutidos.

Feito isso, passamos para a fase de implementação e testes. Após essa etapa, partiremos para o estudo de técnicas de obtenção de QoS em *Web Services*, a fim de promover a incorporação de uma política de escalonamento com diferenciação entre os serviços, fundamental para o atendimento dos requisitos temporais da aplicação.

1.5 Resultados Esperados

Como resultados deste trabalho, esperamos possibilitar uma nova área de aplicação dos *Web Services* como meio de integração de Sistemas Embutidos a outros sistemas, garantindo os requisitos de Qualidade de Serviço impostos por estas aplicações.

1.6 Estrutura do Documento

O restante da dissertação está estruturado da seguinte maneira:

No segundo capítulo é apresentada a revisão da literatura, com os tópicos e conceitos necessários para a compreensão da dissertação.

O terceiro capítulo trata da implantação de *Web Services* em sistemas embutidos, trazendo a discussão sobre o estado da arte, os materiais (plataforma de conectividade e *toolkit* de desenvolvimento) empregados neste trabalho e os trabalhos correlatos. No quarto capítulo é apresentado o modelo proposto para o provimento de *Web Services* em sistemas embutidos e este é exemplificado através de casos de uso.

Depois, no quinto capítulo, é mostrado a realização da implantação e análise dos resultados dos testes do modelo, a fim de que se possa discutir (avaliar e criticar) estes.

Por fim, temos as conclusões deste estudo e a formalização de propostas e perspectivas para trabalhos futuros.

2 *Revisão da Literatura*

Este capítulo aborda os tópicos essenciais para o entendimento do trabalho realizado. Inicialmente abordaremos as principais tecnologias empregadas por *Web Services*. Em seguida definiremos certos conceitos de Qualidade de Serviço em *Web Services* e terminamos esse capítulo com algumas definições sobre Sistemas de Tempo Real e Sistemas Embutidos.

2.1 *Web Services*

Web Services são aplicações fracamente acopladas que interagem dinamicamente através de redes TCP/IP (Internet e/ou Intranets), através da publicação, localização e invocação destes pela *Web* (ROY; RAMANUJAN, 2001), conforme ilustrado na figura 2.1.

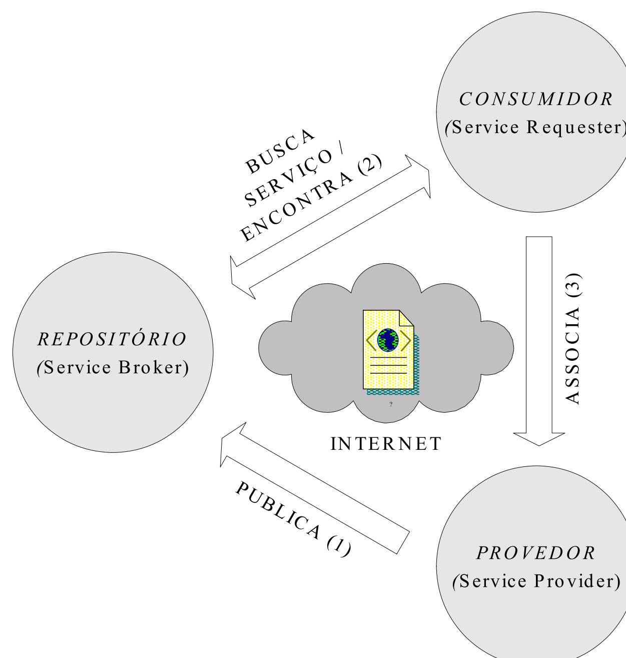


Figura 2.1: Arquitetura SOA

Assim, os serviços disponíveis podem ser descobertos pelos consumidores, possibilitando transações empresariais e comerciais pela rede, sendo este o princípio básico da arquitetura SOA, já mencionada anteriormente.

As principais características de um *web service* (CHAPPELL; JEWELL, 2002) são:

- Baseado em XML;
- Fraco acoplamento entre o consumidor e o provedor do serviço;
- Granularidade grossa;
- Capacidade de uma associação (ligação - *binding*) entre cliente e serviço de forma síncrona ou assíncrona;
- Suporte a chamada de procedimento remoto (RPC - *Remote Procedure Call*);
- Suporte a troca de documentos.

A figura 2.2 ilustra a interação entre os consumidores, provedores e *brokers* (intermediadores) de serviços. Um provedor de serviço registra a descrição de seus serviços através de um arquivo WSDL em um repositório UDDI (1); após, quando um consumidor em potencial (usuário e/ou *web service*) que está procurando por um serviço (2) com as características descritas no WSDL acima mencionado as encontra, é realizado o envio deste arquivo para este cliente. Deste momento em diante, consumidor e provedor são capazes de interagir através de requisições (3) e respostas (4). É a partir dessas características que buscaremos realizar os objetivos deste trabalho.

2.1.1 Principais Tecnologias Empregadas por *Web Services*

Sabendo que *Web Services* não se trata de uma tecnologia específica, e sim um conjunto de protocolos de comunicação e interoperabilidade (consolidados e/ou emergentes) (KILGORE, 2002), conforme ilustrado na figura 2.3, descrevemos as principais tecnologias empregadas pelos *Web Services*.

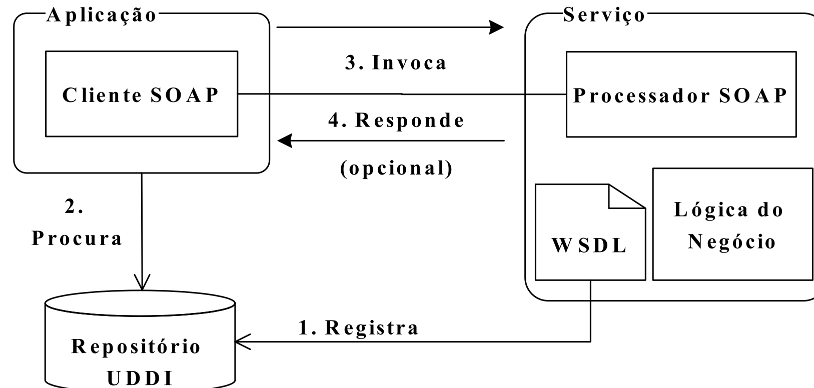


Figura 2.2: Exemplo de interação entre as tecnologias de um *Web Service*.

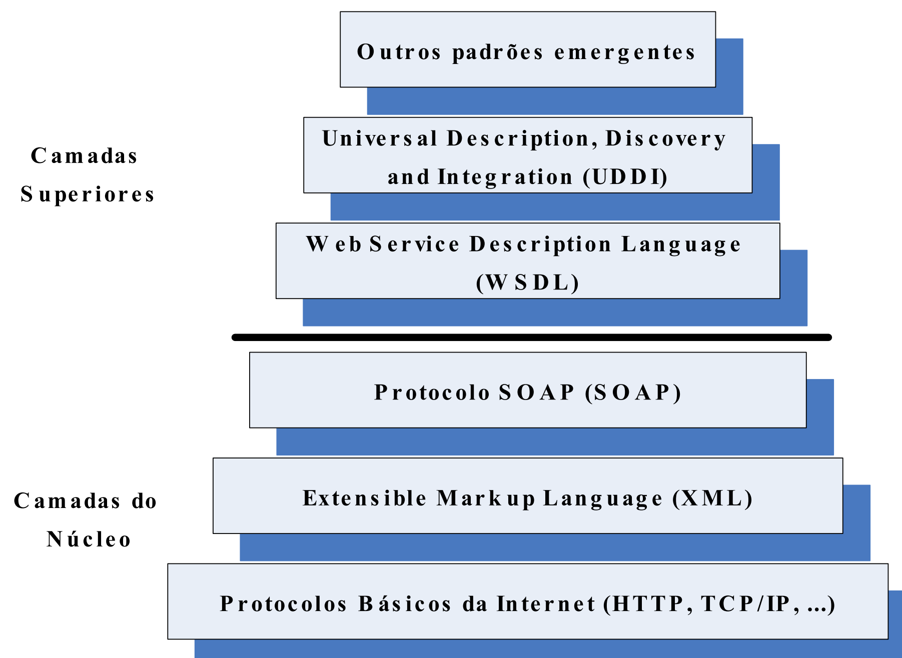


Figura 2.3: Camadas da arquitetura de *Web Service*.

O núcleo de um *Web Service* é composto de um protocolo de Internet (HTTP, geralmente), através do qual são enviadas mensagens XML envelopadas, isto é, devidamente encapsuladas pelo protocolo SOAP. Como camadas superiores (auxiliares, com o intuito de agregar funcionalidades aos *Web Services*) destacamos a linguagem de descrição dos serviços providos pelo *web service* - WSDL e o repositório no qual podem ser publicados e localizados todos os *web services* - UDDI.

Além dessas camadas, existem estudos em andamento, padrões em processo de definição e padrões já definidos, mas ainda não consolidados nos produtos disponíveis atualmente que visam melhorar certas características dos *Web Services*, tais como: Qualidade de Serviço, Políticas de Uso, etc. Apresentados em diversos trabalhos, tais como: (ANDERSON, 2004; GOUSCOS; KALIKAKIS; GEORGIADIS, 2003; TIAN et al., 2003).

2.1.2 XML - *Extensible Markup Language*

A linguagem XML consiste em um padrão definido pelo *World Wide Web Consortium* (W3C) (W3C, 2005a) para especificar e processar as informações (dados). XML é empregada como base de comunicação dos *Web Services* (BONIATI; PADOIN, 2003; LUCCA, 2003). Baseada em TAGs (delimitadores), a linguagem XML descreve os objetos, seus atributos, métodos e parâmetros de forma que os dados sejam interpretados pelas aplicações (CAMELO, 2002).

Um documento XML nada mais é de que um arquivo texto com dados representados em um formato padrão (BONIATI; PADOIN, 2003) no qual é codificada toda a informação contida no documento. A figura 2.4 mostra um exemplo simples de documento XML.

```
<?xml version="1.0"?>
  <Capitulo>
    <Titulo>Revisao Bibliografica</Titulo>
    <Secao>
      <Titulo>Web Services</Titulo>
    </Secao>
  </Capitulo>
```

Figura 2.4: Documento XML.

O documento mostrado na figura 2.4 é composto por três elementos (capítulo, título e seção) definido o conteúdo dos dados (palavras) que ficam entre as marcações - *TAGs*.

Outros dois elementos essenciais em um documento XML e que são definidos à parte são:

- A sua estrutura (tipo de documento e a organização dos elementos), que é definida em esquemas;
- Apresentação (modo com que as informações são apresentadas), feito à parte, através de folhas de estilo, que torna possível que um mesmo dado seja representado de diversas formas.

Portanto existe a necessidade de que os documentos XML sejam bem formados, isto é, estejam de acordo com a recomendação XML 1.0 (W3C, 2005b) e que sejam também válidos para que no momento em que eles forem processados pelo cliente (aplicação) não ocorra nenhum erro.

Esta forma de apresentação/representação é utilizada em uma série de soluções, sendo uma dessas os *Web Services*, que utilizam o XML como base para o envio e/ou recebimento das informações (KILGORE, 2002).

2.1.3 SOAP - *XML Protocol*

Como maneira para facilitar o desenvolvimento de aplicações distribuídas, é desejável que, para a aplicação cliente, a chamada a um método remoto (servidor) seja efetuada de forma transparente, ou seja, não se preocupe com aspectos tais como: alocação de memória, sistema operacional, etc. Deste modo, o cliente pode interagir com o servidor como este fosse um processo rodando localmente. Para isso foi desenvolvido o RPC, e como os *Web Services* também atuam com essa visão (cliente acessa o serviço disponibilizado pelo *Web Service*) foi necessário que o XML pudesse dar suporte a RPCs. Com essa finalidade foi criado o protocolo SOAP, anteriormente conhecido como *Simple Object Access Protocol*.

SOAP é um protocolo leve para a troca de dados XML pela *Web* (ROY; RAMANUJAN, 2001). Serve como um envelopamento de um documento XML para que este possa ser transmitido

pela *Web*. O protocolo SOAP também é o responsável pela codificação dos dados e pelo fornecimento de regras para que possam ser representadas e executadas RPCs nos *Web Services*. Seu envio pela rede pode ser feito usando diversas tecnologias, incluindo SMTP, HTTP, FTP dentre outros. Especificada pelo W3C (W3C, 2005e), uma mensagem SOAP, conforme figura 2.5, é composta por três elementos (BECKER; CLARO; SOBRAL, 2001):

- SOAP *Envelope*, para definição do conteúdo da mensagem e seus *namespaces* (método para que não ocorra o conflito de nomes de elementos, tornando-os únicos na Internet, uma vez que estamos falando de um documento XML, e estes devem ser sempre bem formados e válidos);
- SOAP *Header* (opcional), cabeçalho que contém informações da autenticação, transação e contabilização do *Web Service*;
- SOAP *Body*, que é o corpo da mensagem, no qual estão as informações dos métodos e parâmetros a serem chamados no *Web Service* ou as respostas enviadas por este.

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
        <SOAP:Header>
                <!- Conteúdo do cabeçalho ->
        </SOAP:Header>
        <SOAP:Body>
                <!- Conteúdo do corpo ->
        </SOAP:Body>
</SOAP:Envelope>
```

Figura 2.5: Estrutura de uma mensagem SOAP.
Adaptado de (CURBERA et al., 2002)

Portanto, quando um cliente obtiver a descrição dos serviços oferecidos por determinado *Web Service* através de um repositório, este formatará as mensagens XML e as enviará via SOAP de modo a executar RPCs. Também vale destacar que a partir da versão 1.2 do SOAP é disponibilizado o suporte para comunicação assíncrona, além da síncrona (RPC) previamente mencionada.

2.1.4 WSDL - *Web Service Description Language*

WSDL é uma linguagem utilizada para descrever *Web Services* definida pelo W3C (W3C, 2005c). A descrição em WSDL de um *Web Service* fornece a especificação da interface deste *Web Service*, ou seja, o que cada serviço faz e como invocá-los. É baseado em XML (como o SOAP), e é conceitualmente similar à Linguagem de Definição de Interface (IDL - *Interface Definition Language*) do CORBA (ROY; RAMANUJAN, 2001). A figura 2.6 mostra a estrutura de um documento WSDL (CAMELO, 2002).

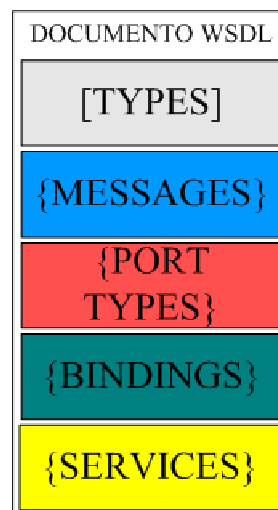


Figura 2.6: Estrutura de um documento WSDL.

Esta descrição inclui detalhes como: definição dos tipos dos dados - *Types*, formatos das mensagens de entrada/saída - *Messages*, operações suportadas pelo serviço - *Port Types*, mapeamento de protocolos - *Bindings* e serviços - *Services* (ROY; RAMANUJAN, 2001). Uma vez definida a descrição do *Web Service* este é obrigado a cumprir o que está especificado em seu WSDL (LUCCA, 2003). Normalmente um arquivo WSDL é gerado automaticamente pela ferramenta de desenvolvimento de software utilizada na criação do *Web Service*. A figura 2.7 apresenta um exemplo de um arquivo que descreve um *Web Service*.

Outro aspecto importante nos documentos WSDL é a possibilidade de geração de código (*stubs*, *skeletons* e parte do código do cliente e/ou servidor) a partir de uma definição WSDL.

```
<types>
<schema targetNamespace="urn:exemplo"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="urn:exemplo"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
</schema>
</types>

<message name="metodoRequest"></message>
<message name="metodoResponse">
<part name="metodoResult"type="xsd:int"/>
</message>

<portType name="exemploPortType">
<operation name="metodo">
<documentation>Service definition of function ns__metodo</documentation>
<input message="tns:metodoRequest"/>
<output message="tns:metodoResponse"/>
</operation>
</portType>

<binding name="chao"type="tns:metodoPortType">
<SOAP:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="metodo">
<SOAP:operation style="rpc"soapAction=/>
<input>
<SOAP:body use="encoded"namespace="urn:exemplo"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<SOAP:body use="encoded"namespace="urn:exemplo"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>

<service name="exemplo">
<documentation>gSOAP 2.7.3 generated service definition</documentation>
<port name="exemplo"binding="tns:exemplo">
<SOAP:address location="http://localhost:80"/>
</port>
</service>
```

Figura 2.7: WSDL

2.1.5 UDDI - *Universal Description, Discovery, and Integration*

O UDDI consiste em um serviço de nomeação e localização de *Web Services* estruturado na forma de repositórios (UDDI, 2005). Após um *web service* ser modelado/desenvolvido é feita a publicação deste num repositório UDDI. A partir desse momento as informações necessárias para a localização e a utilização do serviço disponibilizado por este *web service* se tornam acessíveis para os clientes na forma de um arquivo WSDL.

Na figura 2.8 temos um exemplo de como um *Web Service* (provedor) pode se registrar em um repositório UDDI, com o intuito de disponibilizar a descrição de seus serviços, geradas automaticamente no arquivo WSDL (1), no repositório (2). No instante que um cliente (usuário ou outro *Web Service*) consulta o repositório com o intuito de obter um determinado serviço (3), este recebe o arquivo WSDL do serviço requisitado (4), podendo assim gerar a mensagem apropriada (5) para a utilização deste *Web Service* e transportá-la diretamente para o fornecedor via protocolo identificado pelo WSDL (6), o que caracteriza uma arquitetura SOA citada anteriormente (LUCCA, 2003).

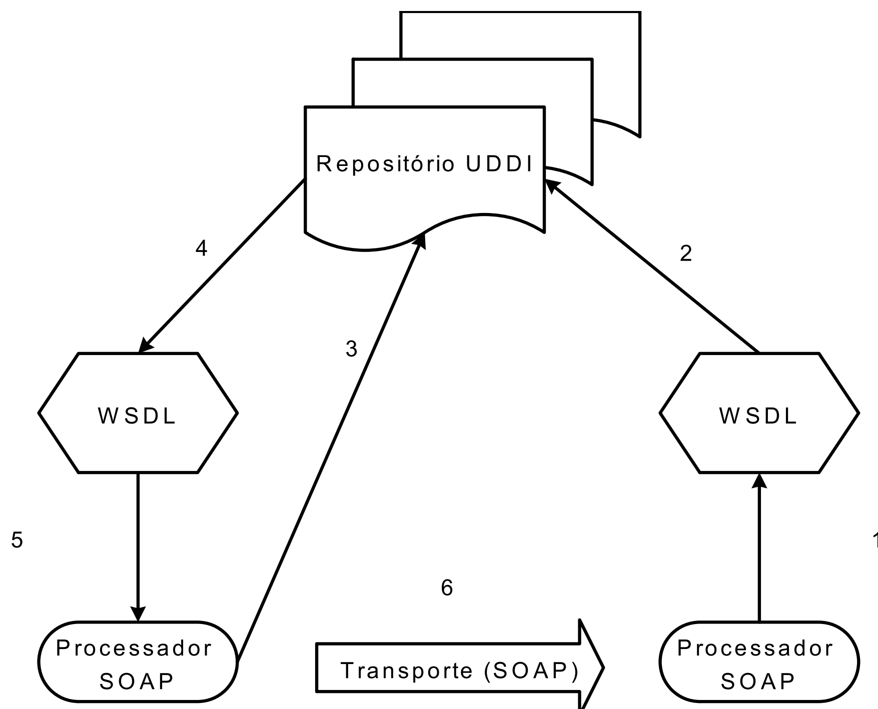


Figura 2.8: Descoberta de *Web Services* através de um repositório UDDI.
Adaptado de (LUCCA, 2003)

Como podem existir diversos repositórios UDDI (KILGORE, 2002), estes devem ser sincronizados para que seus conteúdos sejam idênticos, ou seja, se um *Web Service* é publicado em um repositório A, este é automaticamente replicado para os outros repositórios. Porém essa sincronização não é obrigatória, uma vez que existem casos de instituições/empresas que usam repositórios UDDI privados para proteger seus serviços do acesso externo.

2.2 Qualidade de Serviço em *Web Services*

Conforme visto anteriormente, *Web Services* permitem a interconexão de aplicativos através da rede, e por serem estes sujeitos a restrições impostas pelos consumidores de serviços, é recomendável definir parâmetros de qualidade de serviço (QoS) prestados pelo serviço e pela plataforma.

Web Services ainda não oferecem garantias de QoS, que são requeridas pelos usuários de aplicações corporativas. Estes requisitos podem ser agrupados em três categorias (SIQUEIRA, 2002):

- Requisitos de Confiabilidade: representam a probabilidade de que um determinado serviço consiga ser executado corretamente (ou seja, sem falhas);
- Requisitos de Segurança: indicam que agentes não-autorizados não devem ter acesso a informações confidenciais trocadas pela rede nem ter acesso a serviços os quais não estão autorizados a executar;
- Requisitos Temporais: indicam os limites impostos pelo cliente em relação ao tempo de execução do serviço solicitado.

Sendo que este último é o tipo de requisito abordado neste trabalho.

2.2.1 Especificações de QoS

Qualidade de Serviço (QoS - *Quality of Service*) em sistemas computacionais é a combinação de certas qualidades e propriedades de um serviço (aplicação e/ou rede), isto é, requisitos pré-determinados que este deve cumprir (comportamento esperado) para que se garanta a satisfação do usuário deste serviço.

Alguns parâmetros de QoS a serem considerados pela rede são (SERRA et al., 2004):

- Perda de pacotes;
- Retardo (atraso fim a fim);
- Variação do Retardo (*jitter*).

Já na aplicação, um dos principais parâmetros é o tempo de resposta desta, o que nos leva às seguintes definições (MENASCÉ, 2002):

1. Disponibilidade (*Availability*) que é o tempo de operacionalidade do serviço, isto é, quanto tempo (em porcentagem) que o serviço se mostra adequado, em relação às trocas entre adequado e inadequado;
2. Tempo de Resposta (*Response Time*) é o tempo que o serviço leva para responder aos vários tipos de pedidos (*request*s) por ele aceitos. Pode ser mensurado através de taxas de chegadas de pedidos (pedidos por segundo), pelo número de pedidos processados simultaneamente, ou pode ser considerado em termos de porcentagem (ou seja, quantas pedidos aceitos, em termos percentuais, estão de acordo com um valor ou faixa de valores de tempo de resposta);
3. Vazão (*Troughput*) é a taxa na qual um serviço processa um pedido em um determinado intervalo de tempo.

Web Services têm como meio de comunicação fim a fim a Internet, e esta é baseada na transmissão dos dados através do melhor esforço (*best-effort*). Portanto, a viabilidade de QoS em

Web Services fica dependente da adoção de novos protocolos que operam entre os Clientes *Web* e os Servidores, para suportar tanto o tráfego comum, quanto o tráfego de aplicações que necessitam de QoS (como destaque temos as aplicações multimídia e de tempo real, por exemplo). Soluções como Arquiteturas de Serviços Integrados (*IntServ*) e Serviços Diferenciados (*DiffServ*) apresentam-se como esforços plausíveis para se alcançar QoS, apesar de não estarem disponíveis para toda a rede (CONTI et al., 2002; DIAS; SADOK, 2001). Outra solução disponível é a criação de agentes inteligentes (YUAN, 2003), tanto no lado do cliente, quanto do servidor para que se possa garantir QoS.

Como há a possibilidade de se compor um *web service* utilizando outros *Web Services* , e em alguns casos, já incluindo aspectos de QoS (BENATALLAH; SHENG; DUMAS, 2003; ZENG et al., 2004) é muito importante a análise do tempo de resposta dos serviços que integram o *web service*, pois o impacto de um serviço lento é bastante danoso para o sistema como um todo, principalmente em aplicações que necessitem de garantias de tempo real (MENASCÉ, 2004).[1]

2.3 Conceitos de Sistemas de Tempo Real

Segundo Farines, Fraga e Oliveira (2000), um Sistema de Tempo Real (STR) é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos. STRs possuem algumas características extras em relação à QoS. A principal é que todo Sistema é diretamente afetado pelos requisitos temporais dos resultados obtidos, isto é, não basta oferecer o serviço corretamente (parte lógica), mas também é preciso oferecê-lo no tempo certo (prazos a serem cumpridos) (BURNS; WELLINGS, 2001). Os principais estudos na área buscam soluções em diversos cenários, tais como:

- Controle e Automação Industrial;
- Aplicações Médicas;

[1] Vale ressaltar que desempenho é diferente de tempo real. Em um sistema computacional com restrições temporais, performance média tem pouco significado para o comportamento correto do sistema, logo, ao invés de ser rápido o sistema deve ser previsível.

- Telecomunicações;
- Tráfego Aéreo;
- Multimídia Distribuída.

Já os parâmetros considerados por um STR são os seguintes:

- Período de Ativação;
- Tempo Máximo de Execução;
- Intervalo Mínimo entre Ativações;
- Prazo Máximo de Conclusão da Tarefa (*deadline*), baseado na previsibilidade temporal desta, seja ela determinística (definida no projeto do sistema) ou probabilística (através de uma taxa de erro aceitável).

Como a parte computacional de um STR sempre faz parte de um sistema maior, é interessante estabelecer os 3 componentes que compõem um STR, que segundo Kopetz (1997) são:

- O objeto a ser controlado;
- O sistema computacional;
- O operador desse sistema.

A figura 2.9 mostra a interação destes componentes.

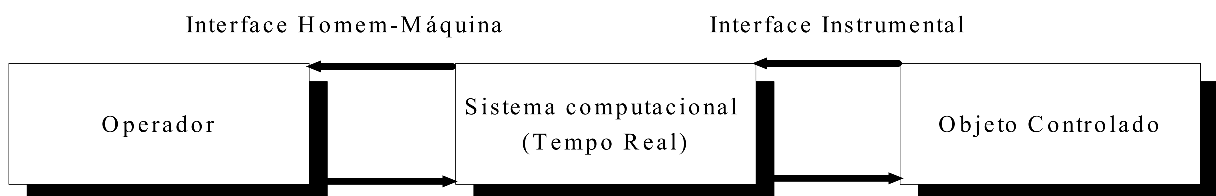


Figura 2.9: Sistema de Tempo Real.
Adaptado de (KOPETZ, 1997)

Além disso os STR são geralmente classificados como *soft real-time systems* e *hard real-time systems.*

SOFT REAL-TIME SYSTEMS são sistemas onde uma falha, ou seja, o não comprimento de um *deadline*, acarreta, provavelmente, num resultado errado. Porém, nestes sistemas, uma falha não se configura em algo crítico para o seu funcionamento. Um exemplo seria um sistema de classificação de objetos (tais como parafusos, rolhas, etc.), se por acaso houver uma falha momentânea e um objeto for classificado erroneamente, e portanto, colocado na caixa errada, não causará grandes prejuízos para empresa.

HARD REAL-TIME SYSTEMS são sistemas onde uma falha, ou seja, o não comprimento de um *deadline*, acarreta, num resultado errado, gerando conseqüências catastróficas. Um dispositivo que controla um equipamento cujo perfeito funcionamento é necessário para manter vivo um paciente se encaixa nessa definição.

2.4 Sistemas Embutidos

Sistemas Embutidos (também conhecidos como Sistemas Embarcados) é uma designação que atualmente perdeu o seu significado original que era de um pequeno sistema computacional, normalmente isolado (*stand-alone*), que dá suporte funcional para dispositivos que não se encaixam na definição de um computador (JANECEK, 2004). Hoje em dia vários dispositivos, principalmente PDAs, são classificados como Sistemas Embutidos.

Podemos definir um Sistema Embutido como um dispositivo microprocessado, portanto programável, que tem como proposta utilizar o seu poder computacional para uma finalidade específica (como por exemplo, decodificadores de TV a cabo, controladores de fornos de microondas, aparelhos de DVD, *chips* de telefones celulares, dentre outros). Logo, estes sistemas possuem características bem peculiares tais como (WOLF, 2001):

- Algoritmos complexos;
- Interface com o usuário específicas (LEDs, *displays*, LCDs, etc...);

- Suporte a tempo real;
- Capacidade de ter várias tarefas rodando ao mesmo tempo em taxas (velocidades) diferentes;
- Baixo custo de fabricação;
- Monitoração e controle do consumo de energia.

O diagrama em blocos de um Sistema Embutido típico é representado pela figura 2.10. Convém lembrar que estes são os componentes mais comuns em um Sistema Embutido, onde todos utilizam algum tipo de memória não-volátil (memória *flash*, EPROM ou ROM, por exemplo) e alguma forma de memória RAM. A maioria também possui alguma interface de comunicação (tais como porta serial, porta Ethernet, etc.) para estabelecer conexões com um *host* de desenvolvimento e/ou algum ambiente de rede.

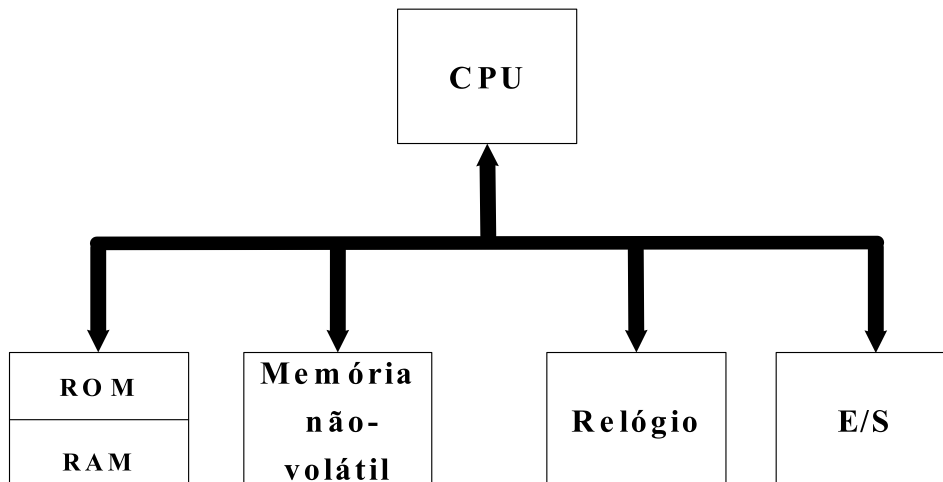


Figura 2.10: Sistema Embutido

Sendo o objetivo principal deste trabalho apresentar a infra-estrutura necessária para permitir o suporte para o desenvolvimento de *Web Services* em sistemas embutidos, mostraremos, nas subseções seguintes, algumas características em relação ao seu S.O. e sua conectividade com a Internet, descritos em (REIS, 2003).

2.4.1 Características dos Sistemas Operacionais para Sistemas Embutidos

Todo projeto de Sistema Embutido depende fortemente do tipo de aplicação e finalidade deste. Deste modo, o sistema operacional (S.O.) utilizado pelo sistema embutido deve seguir vários fatores para a escolha/implementação, uma vez que a utilização de sistemas operacionais de propósito geral, na grande maioria das aplicações embutidas, é inviável em termos de custo. Assim, destacamos os seguintes aspectos como as principais características desejáveis que os sistemas operacionais para sistemas embutidos devem apresentar:

- Lei de Moore, isto é, o S.O. tem que acompanhar a evolução continua de processadores mais complexos e poderosos, sendo capaz de gerenciar tais recursos;
- Custo de licenciamento e disponibilidade do código fonte;
- Suporte a múltiplas arquiteturas de hardware;
- Ferramentas e Suporte de Desenvolvimento;
- Conectividade (sendo este um aspecto que afeta diretamente neste trabalho, pois se o S.O do sistema embutido utilizado não oferecesse suporte à conectividade, não poderíamos utilizar *web services*), melhor discutido na subseção seguinte.

2.4.2 Sistemas Embutidos e a Internet

Os Sistemas Embutidos, cada vez mais, possuem a capacidade de se interconectar com outros equipamentos e dispositivos através de meios e protocolos padronizados. Entre os protocolos suportados pelos Sistemas Embutidos atuais, os protocolos TCP/IP são os que garantem a conectividade destes sistemas com a Internet (e também com as Intranets).

Como vantagens de acessar um Sistema Embutido pela Internet destacamos:

- Interação dos usuários via navegador de Internet;

- Monitoramento, gerenciamento e atualização podem ser feitos de qualquer equipamento através da Internet (o Sistema Embutido tem que oferecer essas funcionalidades);
- Suporte do produto, pelo fabricante, durante todo ciclo de vida;
- Carga remota, no Sistema Embutido, de novas versões das aplicações.

2.5 Considerações Finais

A tecnologia de *Web Services* tem sido empregada de forma a permitir que os sistemas computacionais interajam naturalmente, devido à adoção de meios padronizados de descoberta e uso de serviços que são providos por estes sistemas. Além disso, esta tecnologia utiliza protocolos de comunicação abertos, amplamente disponíveis e um formato de dados padrão, evitando transformações complexas entre protocolos e conversões de dados ao longo do trajeto de comunicação e entre os sistemas alvo.

Os requisitos de Qualidade de Serviço em *Web Services* e sua política de utilização estão em fase de consolidação, e para este trabalho o principal ponto é o aspecto temporal, importatíssimo em Sistemas de Tempo Real e em Sistemas Embutidos.

Grande parte dos sistemas embutidos é capaz de comunicar-se através de uma rede utilizando os protocolos padrões da Internet, como TCP/IP e HTTP, o que nos permite visualizar a possibilidade de utilizarmos *Web Services* para a integração de Sistemas Embutidos a outras plataformas já conectadas à Internet.

3 *Implantando* Web Services *em Sistemas Embutidos*

Conforme visto anteriormente, a arquitetura *Web Services* busca prover formas de integrar aplicações através do uso de padrões, linguagens e protocolos abertos amplamente utilizados na Internet, apesar da heterogeneidade intrínseca deste ambiente distribuído.

Alicerçado em um paradigma que utiliza uma arquitetura orientada a serviços - SOA - baseada em XML, *Web Services* provêem a interconexão de sistemas através de redes TCP/IP, sendo amplamente adotados na integração de aplicações empresariais. No entanto, este tipo de integração ainda não é ofertado no nível de equipamentos baseados em sistemas embutidos, devido à falta de suporte para esta tecnologias na maioria destes dispositivos.

Portanto, definida toda a infra-estrutura e implementados os serviços de acordo com a aplicabilidade e finalidade proposto no projeto do Sistema Embutido, a estrutura de um Sistema Embutido que oferece seu serviços através de *web services* fica de acordo com a figura 3.1.

Neste capítulo são apresentados os requisitos para que possamos prover a infra-estrutura necessária para permitir o suporte para o desenvolvimento de *Web Services* em Sistemas Embutidos.

Discutiremos os equipamentos e *toolkits* disponíveis para implantar *Web Services* em Sistemas Embutidos, apresentando os trabalhos correlatos. Depois comentaremos sobre as principais tecnologias empregadas: a plataforma de conectividade (Sistema Embutido) SHIP e o *toolkit* de desenvolvimento gSOAP.

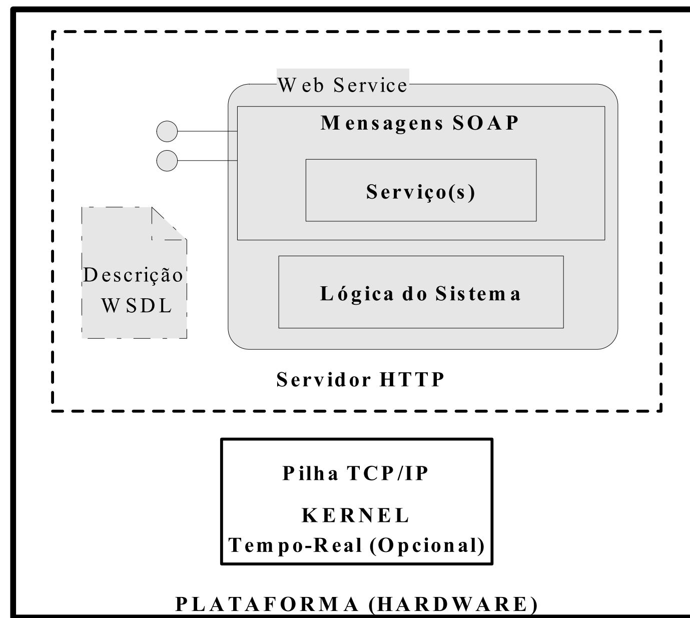


Figura 3.1: Estrutura de um *Web Service* embutido

3.1 Plataformas de conectividade

Foram encontrados na literatura iniciativas acadêmicas e comerciais relacionadas a este trabalho, tais como:

- Netburner (NETBURNER, 2005), plataforma de 32 bits, que utiliza o microcontrolador ColdFire, e que oferece todo um *kit* de hardware, software e ferramentas de desenvolvimento. Possui um S.O. de tempo real, servidor *web* embarcado, pilha TCP/IP e um compilador C/C++, dentre outras características;

- RabbitCore (família RCM) (RABBIT SEMICONDUCTOR, 2005), microcontrolador de 16 bits de fabricação própria, com diversos *kits* de aplicação, oferecendo diversas soluções de conectividade;

- Ethernut (ETHERNUT, 2005), projeto de hardware e software de código aberto para a criação de pequenos dispositivos ethernet embarcados, que utiliza um microcontrolador ATmega128 CPU e uma Realtek RTL8019AS (Ethernut 1) ou LAN91C111 (Ethernut 2) controladora ethernet e possui um S.O de tempo real chamado Nut/OS e uma pilha de protocolos TCP/IP denominada Nut/Net;

- AVR Embedded Internet Toolkit (ATMEL, 2005), plataforma RISC de 8 bits baseada no microcontrolador AVR mega103, é bem apropriado para aplicações embarcadas que necessitem de uma conexão a Internet;
- Picoweb (LIGHTNER Engineering, 2005), utiliza um microcontrolador RISC de 8 bits Atmel AT90S8515, sendo provavelmente um dos menores servidores *web*, que possui todas as funcionalidades básicas, com conexão ethernet;
- iPC - *Internet Processor Chip* (WBC-EUROPE, 2004), sistema embutido construído proveniente da parceria entre Samsung e Thinkware e que já possui o conjunto completo de protocolos (nativos) para *web services* de TCP/IP ao SOAP. Baseado no microcontrolador ARM7TDMI , utiliza um sistema operacional de tempo real (RTOS) chamado Wind River da VxWorks. O iPC tem como principal vantagem a característica de ser uma solução integrada de hardware e software, em relação a implantação de Web Services em Sistemas Embutidos, e como desvantagem podemos citar a pouquíssima documentação;
- MSW (SILVA, 2002), trabalho realizado na UFSC, baseado no microcontrolador Atmel AT90S8515, e que, segundo o autor é capaz "conectar dispositivos elétricos a uma rede padrão Ethernet, baseado em um microcontrolador de baixo custo, utilizando para a comunicação de dados o protocolo TCP/IP e que, permita adquirir, controlar e monitorar remotamente estes dispositivos de maneira segura, eficiente e econômica, mediante o uso de um navegador padrão para Internet";
- SHIP (BORESTE, 2005), que é o ambiente de estudo descrito na seção 3.3.

Existem outras, que apresentam as funcionalidades (pilha TCP/IP nativa e suporte a HTTP) necessárias para integração de dispositivos em ambientes distribuídos através destes sistemas embutidos usando *Web Services*, mas que não são de interesse para este trabalho.

3.2 *Toolkits* de Desenvolvimento

Quanto ao desenvolvimento de *web services* para sistemas embutidos existe uma quantidade razoável de *toolkits*, sejam eles específicos para esta abordagem, ou então que possuem suporte para este tipo de desenvolvimento. Podemos destacar os seguintes *toolkits*:

- Java Platform, Micro Edition - J2ME (SUN Microsystems, 2005), ambiente de desenvolvimento para dispositivos portáteis ou móveis, como celulares e Palms, com uma série de configurações e bibliotecas para estes dispositivos. O suporte a *Web Services* é conseguido através do Java Wireless Toolkit em conjunto com classes que implementam o SOAP e XML, tais como KSOAP (KOBJECTS.ORG, 2006) e KXML (KXML, 2006);
- Microsoft .NET (Microsoft Corporation, 2002), em conjunto com o sistema operacional da Microsoft para pocket PCs, o Windows CE, e através de suas soluções específicas para interoperabilidade - Commom Language Specification e ambientes de múltiplas linguagens - Commom Language Runtime, permitem disponibilizar *Web Services*;
- Emerging Technologies Toolkit - ETTK (IBM, 2005), coleção de tecnologias para o desenvolvimento de software da IBM em diversas áreas emergentes, incluindo *Web Services* e tecnologias XML;
- Fusion (UNICOI Systems, 2005), através do pacote Fusion Web, oferece aos desenvolvedores de aplicações embarcadas soluções para toda a topologia de protocolos e tecnologias envolvidas em *Web Services*;
- eSOAP (EXOR, 2005), biblioteca que abstrai todos os aspectos do SOAP/HTTP, desenvolvedor só precisa se familiarizar com algumas classes do *toolkit* para que o Sistema Embutido utilize a *Web*;
- gSOAP (GENIVIA, 2005), que é a nossa ferramenta de trabalho descrito na seção 3.4.

Como principais restrições para a escolha de um *toolkit* apontamos as seguintes características almejadas:

- Que funcione em hardware de porte limitado (processamento, memória, sistema operacional, etc...);
- Ambiente de desenvolvimento amigável (que não demandem a utilização de ferramentas pagas ou que precisem de uma série de outros softwares - como compiladores específicos - para sua utilização, por exemplo);
- Boa documentação para o usuário/desenvolvedor.

3.3 SHIP

O primeiro passo para a verificação das idéias descritas no trabalho consistiu na seleção da plataforma alvo para implantação de *web services* em sistemas embutidos. Através de um convênio firmado pelo Laboratório de Pesquisas em Sistemas Distribuídos - LaPeSD/INE/UFSC com a empresa Boreste, adquirimos o SHIP (versão de desenvolvimento) (BORESTE, 2005).

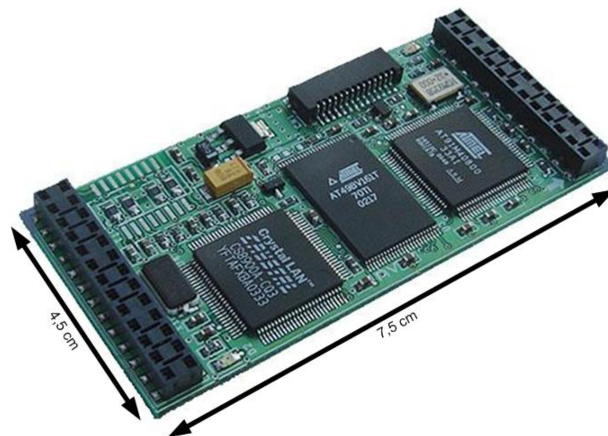


Figura 3.2: SHIP - Software e Hardware Integrados em uma Plataforma

O SHIP é uma plataforma (*socket card*) criada com o intuito de prover conexões Ethernet base 10baseT em redes TCP/IP, isto é, através de aplicações nativas na plataforma (*web services*, por exemplo) é possível receber e transmitir dados. O SHIP possui as seguintes características:

- Microcontrolador ARM7TDMI, da família AT91X40 com processador RISC 32 bits de 20MHz;

- Memória flash de 512K Bytes, sendo 448K Bytes livres para as aplicações;
- Sistema Operacional desenvolvido pela Boreste estruturado para dar suporte a comunicações TCP/IP sobre uma rede ethernet. Tem como principais características o μBoot para carga do sistema e um programa monitor - μMonitor - para configuração e carga das aplicações na plataforma;
- Suporte as rotinas padrões em C, tais como: `stdlib`, `stdio`, `ctype`, `string`, etc;
- Servidor compacto de páginas *Web* (também desenvolvido pela empresa);
- Diversos dispositivos externos podem ser ligados à plataforma desde equipamentos de automação da manufatura, passando por sensores e atuadores, equipamentos médicos, etc...

3.4 gSOAP

Em um segundo momento, mostrou-se necessário portar um *toolkit* de desenvolvimento para a plataforma alvo. Optamos pelo *toolkit* gSOAP (GENIVIA, 2005), que se mostrou o mais compatível em relação ao suporte de hardware/software da plataforma.

Idealizado pelo Professor Robert Van Engelen na Florida State University, o *toolkit* gSOAP é capaz de criar aplicações *web services* e clientes via C e C++. Como mencionado em (ENGELEN; GALLIVAN, 2002), gSOAP provê total interoperabilidade com o protocolo SOAP.

Este *toolkit* possui as seguintes características:

- Faz a composição (*binding*) de mensagens SOAP com C/C++ criando *Stubs* e *Skeletons*;
- Cria clientes C/C++ a partir da descrição WSDL;
- É independente de plataforma (possui exemplos de aplicações desenvolvidas em Windows, Linux, Unix, Mac OS X, Pocket PC, Palm OS, Symbian e embedded Linux);

- Aplicações podem ser criadas com menos de 100K Bytes, totalizando um consumo total de memória de 150K Bytes.

A figura 3.3 descreve os estágios para o desenvolvimento e disponibilização de um aplicação (*web service*) com o *toolkit* gSOAP, esses estágios, bem como uma série de características auxiliares desse *toolkit* são apresentados em (ENGELEN; GUPTA; PANT, 2003).

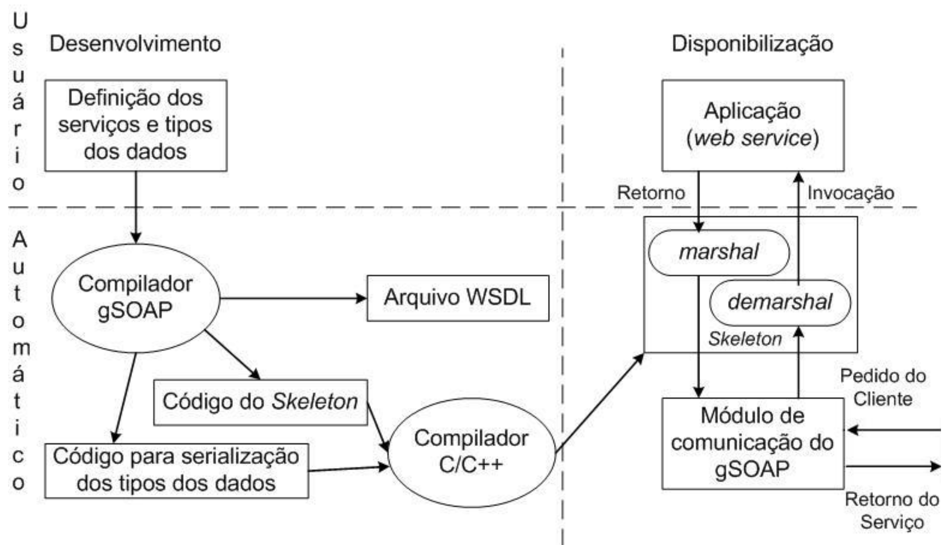


Figura 3.3: Desenvolvendo e Disponibilizando um serviço com gSOAP

De forma análoga temos a descrição dos estágios para o desenvolvimento e disponibilização de um cliente gSOAP, conforme figura 3.4.

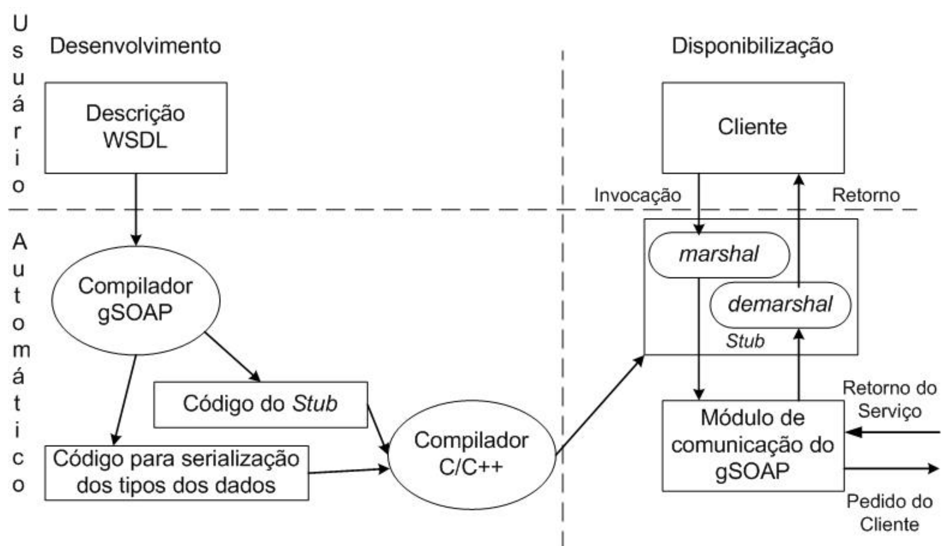


Figura 3.4: Desenvolvendo e Disponibilizando um cliente com gSOAP

Vale destacar que não é necessário a criação de um cliente através do gSOAP para o acesso de uma aplicação que também foi criada com o gSOAP, ou seja, podemos utilizar outros *toolkits* de desenvolvimento e até outras formas de acesso (formulários web, por exemplo).

Um detalhe relevante para a escolha do gSOAP como *toolkit* de desenvolvimento utilizado neste trabalho é que, além de gerar a composição de mensagens na linguagem C (ítem indispensável, uma vez que esta é a única linguagem suportada pelas bibliotecas de desenvolvimento do SHIP), conforme demonstrado em (GOVINDARAJU et al., 2004), o desempenho do gSOAP, baseado em um trabalho anterior (ENGELEN, 2003), é o melhor dentre uma série de *toolkits* para desenvolvimento de *Web Services*, o que é fundamental quando o ambiente a ser utilizado (plataforma embutida) exige essa característica.

Outro ponto a favor desse *toolkit* é que, segundo documentação presente no site do fabricante, bem como em análises de outras empresas/usuários que o utilizam, é que o gSOAP é o *toolkit* de melhor interoperabilidade com os outros *toolkits* disponíveis para o desenvolvimento de *Web Services*, principalmente por utilizar fielmente todos os padrões propostos pelo W3C. Além disso, possui uma farta documentação, bem como uma lista de discussão com bastante atividade.

Por outro lado, em (ENGELEN, 2004) é descrito o uso do *toolkit* gSOAP para o desenvolvimento de *web services* em plataformas embutidas com maior capacidade, tais como *Personal Digital Assistants* (PDAs), e também cita o gSOAP como parte dos pacotes de software para desenvolvimento em sistemas embutidos.

Suas distribuições mais atuais possuem módulos de integração com o servidor Apache, por exemplo, além de várias características como conexão segura, bibliotecas prontas para se trabalhar com Palm, etc. A codificação e depuração dos servidores e clientes feitos com esse *toolkit* é bem fácil para quem possui uma certa familiaridade com C/C++.

Usando o compilador do *toolkit* (soapcpp2) foi obtida uma ligação transparente entre os tipos de dados do C e C++ e os tipos de dados do SOAP/XML. Os arquivos gerados contém os *stubs* e *skeletons* necessários para enviar mensagens através do protocolo SOAP.

3.5 Porte, adaptação do *Toolkit* para a Plataforma

Para realizarmos a implantação de *Web Services* (utilizando o *toolkit* gSOAP) em sistemas embutidos (plataforma SHIP, neste trabalho), fizemos certos ajustes entre a plataforma e o *toolkit*.

Foram detectadas e realizadas as alterações necessárias no *firmware* da plataforma para possibilitar a disponibilização de *Web Services*. Para isso foi realizado o porte (compilação) da biblioteca gSOAP, criando assim uma API compatível com o gSOAP (`libgsoap` - sendo que o `makefile` para criação desta biblioteca está disponível no anexo IV deste documento) e adaptação da camada *sockets*, pois esta apresentava problemas com a liberação de memória, onde persistiam lacunas não utilizadas (que não eram reagrupadas) e no decorrer de uma sessão comprometiam a alocação de blocos maiores. O *firmware* do controlador Ethernet da plataforma SHIP precisou ser revisto a fim de permitir que requisições possam ser despachadas com um tempo de resposta razoável e que este pudesse suportar uma grande quantidade de requisições simultâneas sem que houvesse o estouro do *buffer* de requisições. Com os aprimoramentos efetivados a latência no acesso à rede foi reduzida em cerca de 85%.

A plataforma (SHIP) adotada não possui recursos suficientes que viabilizem a disponibilização de um servidor *web* completo como Apache (Apache, 1996) e IIS (Microsoft, 2005). Deste modo, foi utilizada a capacidade de disponibilização de *Web Services* no modo *stand-alone* disponível no gSOAP, ou seja, a própria biblioteca é capaz de gerenciar requisições HTTP. Essa solução é conhecida como SaaS (*Software as a Service*).

Outro ponto crucial é que, por enquanto, não temos uma biblioteca para gerenciamento de *threads* no S.O. do SHIP (atualmente segue um modelo de software orientado a eventos), logo os serviços são executados a partir de uma única *thread* e de forma não preemptiva. Outro detalhe importante é que o S.O. do SHIP não disponibiliza um sistema de arquivos, portanto não é possível guardar certos dados (como por exemplo, um arquivo WSDL), porém existe a possibilidade de utilizarmos o servidor compacto de páginas Web desenvolvido pela Boreste, que disponibiliza as páginas HTML no próprio código do servidor, ou seja, o arquivo WSDL

poderia ser mantido em memória e provido pelo servidor HTTP.

Vale destacar que o SHIP mostrou-se plenamente satisfatório, em relação à capacidade de memória e processamento para que realizássemos o *parsing* do XML (dados e parâmetros).

Foi necessário realizar algumas modificações no código fonte de modo que o gSOAP ficasse compatível com o sistema operacional do SHIP. Algumas definições do gSOAP nos arquivos `configure.h`, `soapdefs.h` e `stdsoap2.h` foram modificadas para se ajustarem à capacidade do SHIP. Complementando, algumas questões de compatibilidade a respeito das constantes GNU libc também foram mudadas para valores de acordo com a glibc-2.2.

O arquivo `config.h` (que é gerado pelo comando `./configure`) foi editado afim de alterarmos os `#defines` que identificam as capacidades do ambiente para que este se ajuste à plataforma. Depois disso, alterou-se também o `soapdefs.h` e o `stdsoap2.h`, porque existiam definições nestes arquivos que não são configuradas pelo `./configure`.

Além disso, o autor assume certas constantes a partir da GNU libc, o que não é sempre verdadeiro para todas as bibliotecas C do mundo. Por exemplo, ao invés de usar os `defines` SHUT_RD, SHUT_WR e SHUT_RDWR, ele usa os valores 0, 1 e 2. Esse tipo de mudança é complicada de procurar e corrigir, mas é imperativo que se faça, pois gera um tipo de *bug* muito difícil de detectar. Disponibilizamos no anexo III, os arquivos `config.h`, `soapdefs.h e stdsoap2.h.patch`, que é um 'diff' do `stdsoap2.h` utilizado neste trabalho em relação ao original (que é bem extenso) e serve como guia do que no `stdsoap2.h`.

A versão do gSOAP portada para plataforma é a 2.7.3 e este processo não é automatizado, logo a tentativa de compilar *web services* com outras versões não é possível, sendo que há ainda alguns problemas quanto às mensagens de erro geradas pelo servidor e que, caso os parâmetros dos serviços criados sejam números, estes devem ser valores inteiros, uma vez que o SHIP não trabalha com pontos flutuantes.

Convém lembrar que o SHIP tem um processador modesto, uma interface de rede de baixa velocidade e pouca memória RAM (512kB, sendo que o servidor de aplicação gerado pelo gSOAP executando um web service básico consome aproximadamente 65KBytes de memória.

Cada serviço adicional consome cerca de 2KBytes de memória, variando conforme a complexidade do serviço).

3.6 Considerações Finais

A implantação de *Web Services* em plataformas embutidas não só é possível, como já existem os primeiros esforços nesse sentindo.

Dentre uma série de fatores optamos pela utilização do par SHIP (plataforma de conectividade ethernet) + gSOAP (*toolkit* de desenvolvimento), o que, após os ajustes necessários para o bom funcionamento deste par, se mostrou plenamente satisfatório. Isso possibilitou definirmos um modelo de arquitetura para o provimento de *Web Services* em Sistemas Embutidos, bem como de aplicarmos técnicas de classificação de serviços oferecidos nesse modelo.

4 *Diferenciação de Serviços em Web Services*

Neste capitulo apresentamos a infraestrutura criada para o desenvolvimento de *Web Services* em Sistemas Embutidos com suporte a classificação entre os serviços.

Hoje em dia, grande parte dos sistemas embutidos é capaz de comunicar-se através de uma rede utilizando os protocolos padrões da Internet, como TCP/IP e HTTP. Baseado no suporte provido por estes protocolos e utilizando um *toolkit* para desenvolvimento de *web services* compatível com os sistemas embutidos, propusemos uma arquitetura (Figura 4.1) para a integração de dispositivos embutidos baseado em *Web Services* (MACHADO et al., 2006).

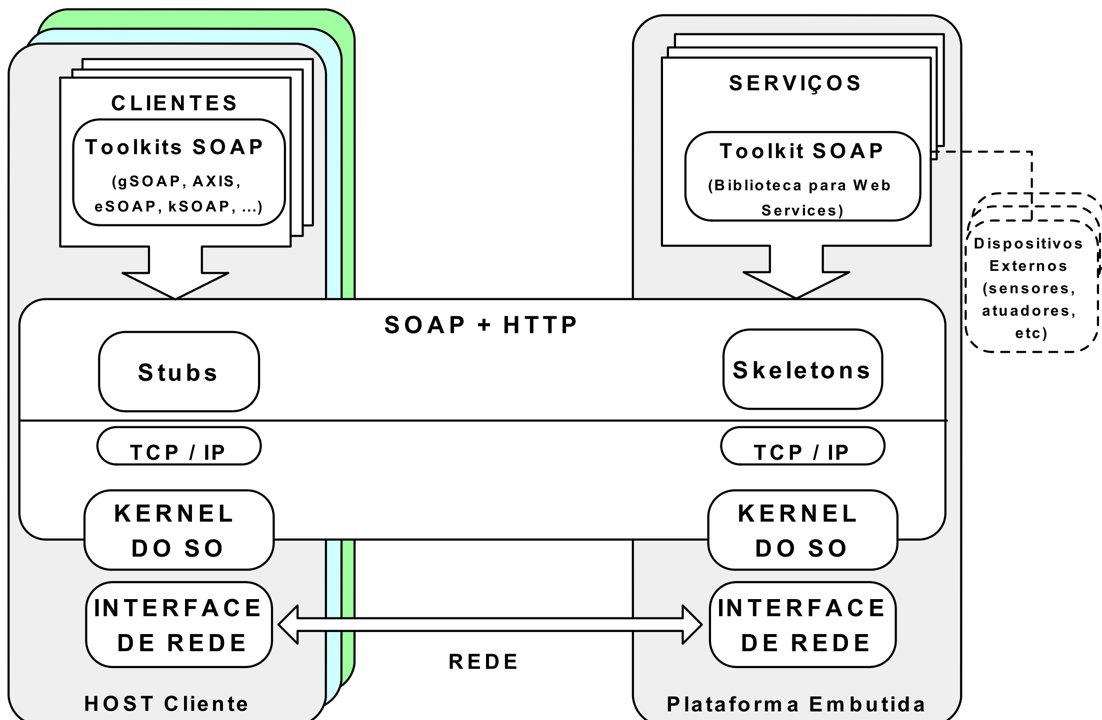


Figura 4.1: Arquitetura do Ambiente

Portanto, a arquitetura proposta é baseada em plataformas embutidas com sistemas operacionais que provêem suporte à pilha TCP/IP. Além disso, o *toolkit* de desenvolvimento escolhido teve que ser portado para a plataforma, permitindo criar os *skeletons* que irão serializar e desserializar as mensagens SOAP. Similarmente a plataforma cliente deverá prover suporte à TCP/IP, permitindo que os *stubs* (criados por qualquer tipo de *toolkit* SOAP utilizado neste cliente) interajam com a plataforma embutida enviando/recebendo mensagens SOAP via protocolo HTTP.

Neste ambiente, um ou mais *hosts* possuem um ou mais clientes que enviam suas mensagens XML/SOAP de acordo com o tipo de serviço requisitado (previamente conhecido através de um arquivo WSDL localizado na plataforma embutida e/ou em um repositório UDDI) pelo protocolo HTTP.

Esta mensagem chega à biblioteca para Web Services empregada na construção do serviço e, então, o método chamado pelo cliente será processado pelo microcontrolador do dispositivo. Após a execução, que poderá ocorrer na plataforma ou em dispositivos externos (tais como sensores e atuadores) conectados a esta, uma resposta gerada pelo serviço poderá ser retornada para o cliente.

4.1 Solução

Uma solução para oferecer QoS em aplicações em que o tempo de resposta é o principal fator de qualidade do sistema é a utilização de Diferenciação de Serviços. Com *DiffServ* os serviços são classificados em classes onde cada tipo tem uma maior ou menor prioridade na alocação dos recursos (tanto no nível do sistema como para o nível da aplicação, exemplificado em (RYU; KIM; HONG, 2001)).

Uma proposta para que se aplique QoS em *Web Services* com requisitos de Tempo Real é oferecer diferentes níveis de prioridade para cada serviço específico do *Web Service* (SERRA et al., 2004), ou que seja criado um mecanismo inteligente (SHARMA; ADARKAR; SENGUPTA, 2003) através de priorização estática e dinâmica das requisições. Com isso, serviços que necessitem

de respostas em tempo real (sinais de controle e monitoramento de um avião, por exemplo) terão as melhores taxas de alocação dos recursos, tais como: menor *jitter*, preferência na alocação de memória, maior prioridade na fila de escalonamento de processos, menor tempo de resposta, entre outros.

A implementação desta proposta segundo Sonda e Montez (2004), implica em três atividades:

- Definição de classes de serviços;
- Mapeamento (classificação) de requisições para um serviço para uma determinada classe;
- Definição de políticas que implementem essa diferenciação de serviços.

Com isso poderemos utilizar mecanismos de adaptação de acordo com cada serviço, isto é, dependendo do contexto em que esse serviço é chamado, ou durante a sua execução ele poderá sofrer as seguintes alterações (CERVIERI; OLIVEIRA; GEYER, 2002), e de acordo com nossa proposta (Seção 4.2):

- Atrasar a conclusão da tarefa;
- Cancelar a execução de uma tarefa;
- Variar o tempo de execução da tarefa.

O escalonamento das tarefas seguirá a abordagem da figura 4.2.

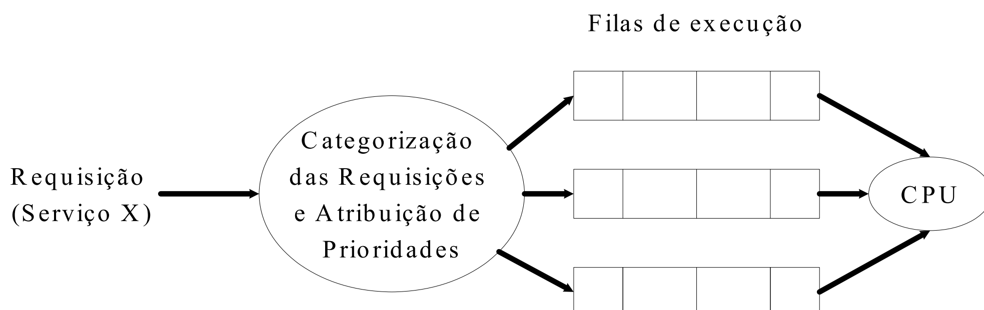


Figura 4.2: Abordagem de escalonamento

Essa abordagem pode gerar o problema de *Starvation*, ou seja processos de menor prioridade podem nunca ser executados, uma solução para esta situação seria a delegação de um tempo máximo de espera em cada fila de execução.

Apesar de termos definido certas especificações de QoS na subseção 2.2.1, convém lembrar que não faz parte do escopo deste trabalho definir os aspectos de QoS providos pela rede, tampouco implantar as alterações definidas acima (atrasar a conclusão, cancelar ou variar o tempo de execução de uma tarefa). O que desejamos é definir e escalonar os serviços de acordo com as Classes de Serviços propostas neste trabalho.

Vale a pena lembrar que hoje em dia já existem soluções de escalonadores, configuráveis, de tempo real modelados em hardware (KUACHAROEN; SHALAN; MOONEY III, 2003), o que poderia ser uma solução para os assuntos discutidos nos parágrafos anteriores.

Os procedimentos abaixo, demonstram como aplicamos a nossa abordagem.

Procedimento 1 Definição e Escalonamento dos Serviços - Função principal

---

1. Aguardar conexão;
2. QUANDO houver pedido de interrupção; // Tentativa de conexão
3. SE valor do pedido de interrupção = -1:
   (a) erro de conexão;
4. SE valor do pedido de interrupção = 0:
   (a) timeout;
5. SE não existir conexão válida então:
   (a) criar conexão;
   (b) servidorsoap_enfilere(método);
6. SENÃO: //há um pedido de interrupção E $\in$ conexão
   (a) servidorsoap_enfilere(método);
7. processe_requisição(método);
8. servidorsoap_desinfilere();

Procedimento 2 Definição e Escalonamento dos Serviços - Filas de execução

1. servidorsoap_comece;
2. SWITCH (método->classe);
3. Caso 1:
   (a) inserir_na_lista(&gold, método);
4. Caso 2:
   (a) inserir_na_lista(&silver, método);
5. Caso 3:
   (a) inserir_na_lista(&bronze, método);
6. padrão:
   (a) inserir_na_lista(&best, método);
7. retorne;

Para isso efetuamos uma extensão no gSOAP, que nos possibilita trabalharmos com esse abordagem de classificação. Essa extensão funciona como um *patch*, que substitui o arquivo `soapServer` gerado automaticamente pelo compilador do gSOAP, por outro, criado manualmente, onde cada método recebe a sua categorização e que modifica a estrutura interna da biblioteca gSOAP, permitindo que esta aceite (enfilere) mais de uma conexão, ou seja não bloqueie o servidor enquanto este esta processando uma requisição.

4.2 Características da Abordagem Proposta

Um vez que o sistema embutido usado como plataforma de estudo/utilização possui funcionalidades que possibilitam o armazenamento ou aquisição de dados e a troca destes via protocolos Internet, existe a possibilidade da disponibilização de *web services* (serviços) neste dispositivo.

A proposta inicial é de disponibilizar uma descrição dos serviços disponíveis no *web service* via WSDL e que existam diferentes classes de serviço, para que possamos trabalhar aspectos

de qualidade de serviço (QoS) e processamento em tempo real. Uma vez definidos os serviços e a descrição destes, a elaboração da aplicação (cliente) pode ser feita de acordo com as necessidades e recursos do usuário, podendo ser uma página, outro sistema embutido, um programa, ou outros recursos.

Basicamente, o sistema proposto segue a visão da figura 4.3, no qual um cliente envia suas mensagens XML envelopadas com SOAP de acordo com o tipo de serviço requerido (previamente conhecido através do arquivo WSDL localizado no próprio Sistema Embutido e/ou em um repositório UDDI) através do protocolo HTTP.

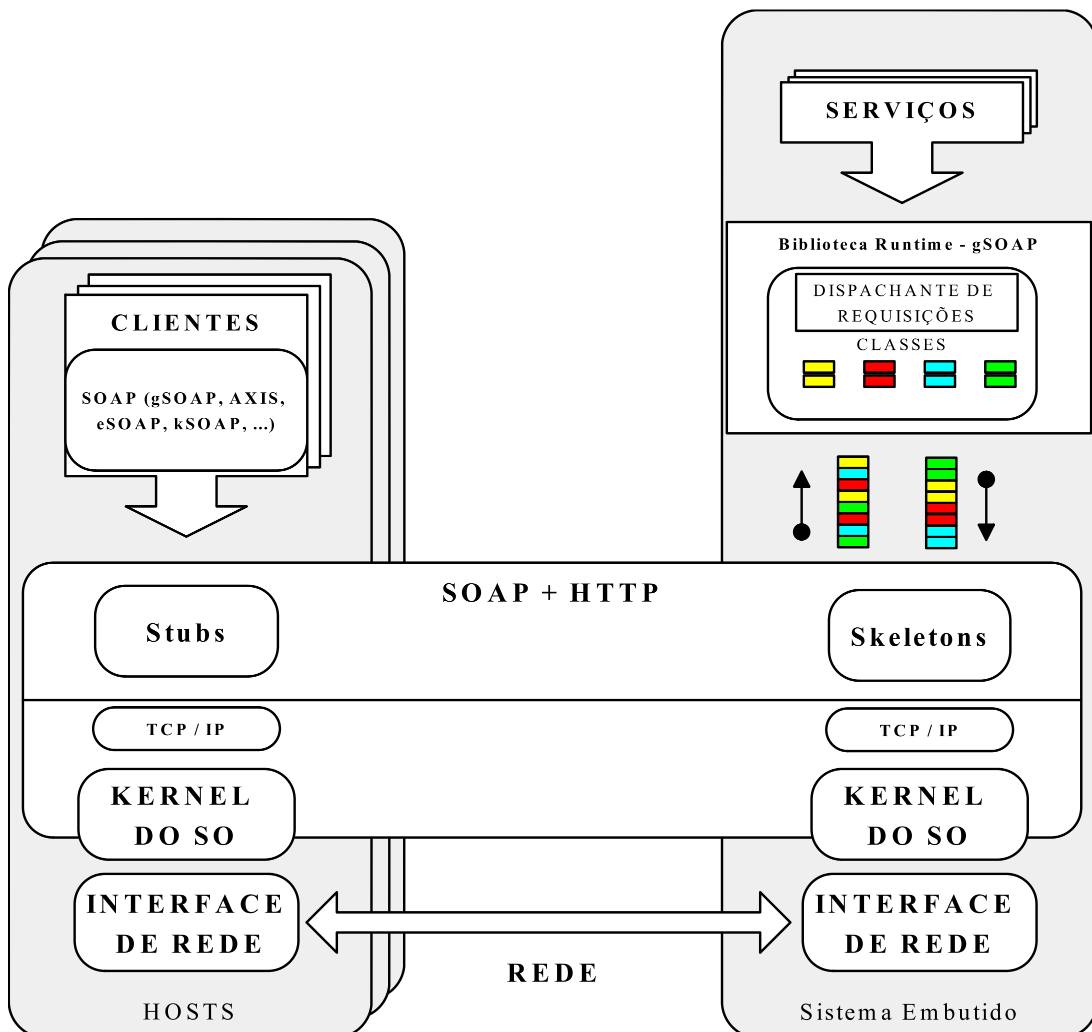


Figura 4.3: Sistema proposto.

No *web service* há um processamento (categorização, seguindo a metodologia proposta discutida na subseção 4.1, que consiste na distribuição eficiente dos serviços, ou seja, prioridades

de escalonamento, baseado na diferenciação - *gold*, *silver*, *bronze* e *best effort* - destes) dessas mensagens que serão então processadas pelo microcontrolador. Feita a execução das requisições contidas nestas mensagens, que pode ser uma resposta do próprio microcontrolador ou do dispositivo remoto ligado a ele, estas são retornadas ao cliente.

4.3 Considerações Finais

Ao investigarmos uma metodologia capaz de provermos uma arquitetura para que se possa disponibilizar *Web Services* em Sistemas Embutidos com classificação dos métodos dos serviços empregados por uma plataforma embutida, chega-se a uma abordagem que se mostrou plenamente aplicável através da implantação e análise dos resultados dos testes, baseados no cenários de utilização do modelo proposto descritos no próximo capítulo.

5 *Implantação e Análise dos Resultados*

Neste trabalho (baseado no paradigma *software as a service* - SaaS) foram construídas aplicações cliente-servidor utilizando a plataforma para executar um serviço e PCs como clientes, demonstrando assim a possibilidade da execução de *web services* em sistemas embutidos.

O desenvolvimento das aplicações para a plataforma SHIP é realizado no ambiente Cygwin (CYGWIN, 2005) configurado com o *cross compiler* ARM GNU *toolchain* (Macraigor Systems, 2005), com o kit de desenvolvimento (bibliotecas e exemplos) da plataforma e com o *toolkit* gSOAP (versão adaptada à plataforma). A carga das aplicações é efetuada a partir da interface serial do SHIP controlada pelo programa monitor da plataforma, acessado por um emulador de terminal (Hyperterminal do Windows, por exemplo).

Um breve tutorial para a configuração, instalação e utilização do gSOAP esta disponível no anexo I deste documento, bem como um exemplo de aplicação (anexo II).

As figuras abaixo descrevem como é realizado o processo de implantação do modelo proposto. Em um primeiro momento deve-se criar todo o ambiente de execução, de acordo com a figura 5.1. O processo, através de um arquivo de `makefile` (em anexo IV neste documento), funciona da seguinte forma:

- *Parsing*, para a criação da definição dos serviços e tipos de dados, através da descrição WSDL do *Web Service* a ser criado;
- Compilação do arquivo gerado pelo *parsing* para a criação, em C, dos códigos (arquivos) para a serialização dos tipos de dados, *stubs* e *skeletons* do ambiente;

- Verificação se utilizamos a extensão (classificador) do gSOAP;

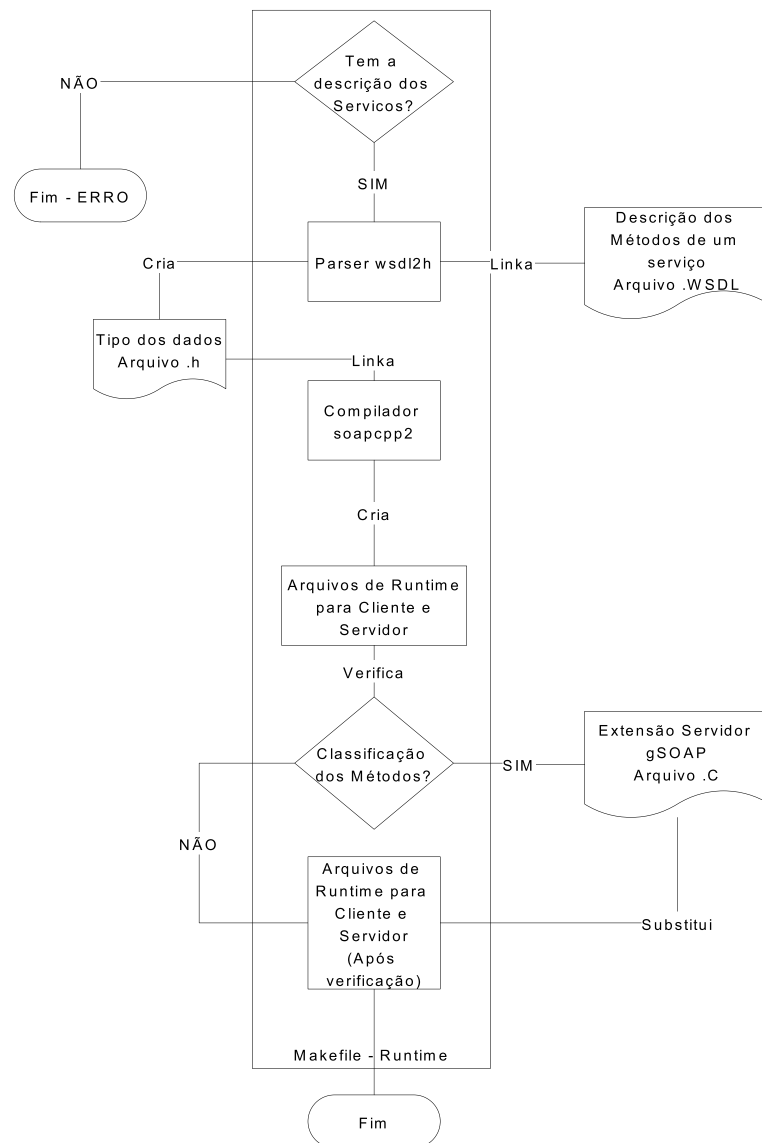


Figura 5.1: Criação do Ambiente de Execução - com ou sem classificação do métodos

Terminada a criação do ambiente de execução, partimos para a disponibilização do *Web Service* (servidor) na plataforma. O processo se dá de acordo com a figura 5.2, onde compilamos, os arquivos de ambiente, a biblioteca gSOAP portada para nossa plataforma, os *header files* e o servidor (aplicação) propriamente dito, afim de que possamos gerar o binário que será carregado na plataforma. O makefile que automatiza esse processo está no anexo IV deste documento.

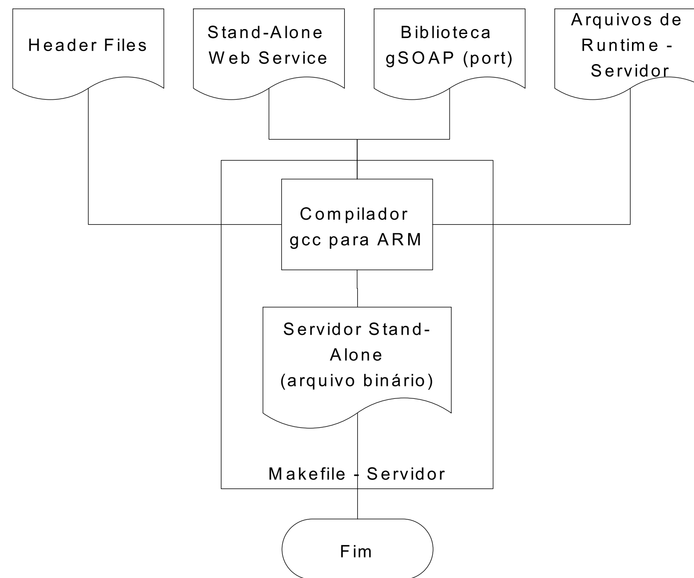


Figura 5.2: Compilação do Web Service

5.1 Casos de Uso

Nesta seção descrevemos dois cenários de utilização onde se aplicaria as idéias da nossa abordagem proposta, e nos quais nos baseamos para a implantação dos serviços analisados.

5.1.1 Integração de um Sistema de Controle de Redes Industriais

A arquitetura proposta teve sua aplicabilidade verificada na definição de um cenário experimental para integração de um sistema de controle de redes industriais.

De acordo com a figura 5.3, a arquitetura proposta pode prover o suporte para a integração de redes industriais heterogêneas, tais como a *Controlled Area Networks* (CAN) (BOSH, 1991) e PROFIBUS (PROFIBUS, 2002). Os elementos de integração (a plataforma embutida) funcionam como um conversor de protocolo para cada barramento de rede e, para os outros dispositivos conectados na rede ethernet, cada equipamento industrial será visto como um *Web Service*. Portanto, não será mais necessário utilizar um PC para supervisionamento do controle e aquisição dos dados para cada rede na fábrica, aumentando a confiança de funcionamento, reduzindo gargalos e também eliminando a necessidade de softwares customizados - e caros - para integração.

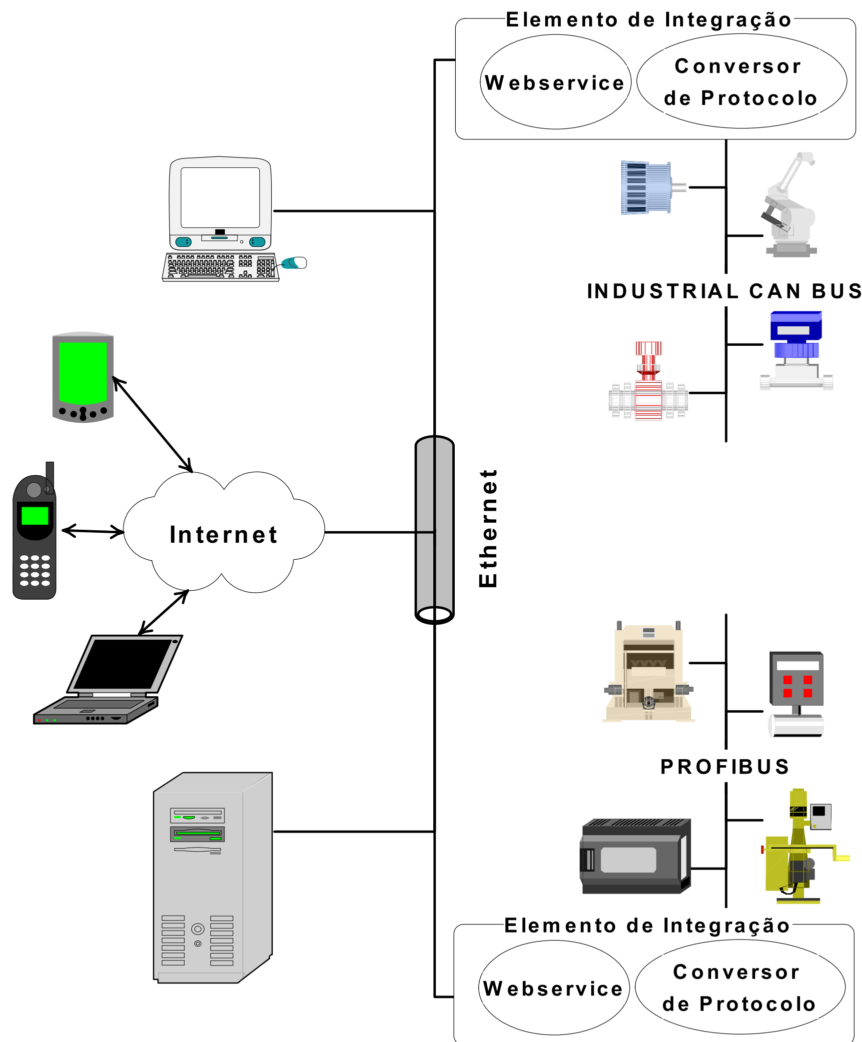


Figura 5.3: Cenário de Integração em Sistemas de Controle

Portanto, neste cenário, mensagens SOAP serão trocadas entre o elemento de integração e seus clientes, enquanto dentro de cada rede industrial as mensagens serão trocadas através do formato específico de cada rede. Cada rede implementa seus próprios mecanismos de controle, permitindo restrições de tempo impostas por cada célula de produção e satisfazendo as restrições temporais deste ambiente de modo determinista.

Apenas mensagens de controle de alto nível (tais como *start*, *stop* e *status)* serão trocadas entre os elementos de integração e seus clientes, e sempre que possível estas mensagens serão definidas em arquivos WSDL para cada *web service* utilizado na fábrica. Se desejarmos, os métodos destas mensagens podem ser classificados, ou seja, receberem maior ou menor priori-

dade de escalonamento de acordo com a relevância de cada método. Os métodos responsáveis por iniciar ou parar uma máquina, por exemplo, seriam classificados como de maior importância - *gold* na nossa abordagem - do que o método que verifica o estado desta máquina.

As figuras 5.4 e 5.5 demonstram como, a partir de definições (em C) dos tipos dos dados utilizados pelas mensagens descritas acima, podemos gerar automaticamente - através de um *toolkit* para desenvolvimento de *web services* (gSOAP neste caso) - um arquivo WSDL que representa a descrição dos serviços a serem utilizados.

```
//gsoap ns service name:  chao
//gsoap ns service name:  chao
//gsoap ns service style:  rpc
//gsoap ns service encoding:  encoded
//gsoap ns schema namespace:  urn:chao
int ns__start(int *StatusResult);
int ns__stop(int *StatusResult);
int ns__status(int *StatusResult);
```

Figura 5.4: Definições em C

Uma outra hipótese de abordagem seria a de utilizarmos *web services* até a rede ethernet deste cenário de utilização, fazendo com que o elemento de integração atue como uma ponte, ou seja, um *gateway* entre o protocolo local de cada rede (CAN, Profibus, etc) e uma solução CORBA ou RMI que se comunica com a rede ethernet, evitando que seja enviada uma mensagem com tamanho muito grande até este, pois poderiam existir situações onde um único bit seria necessário para ativar certo equipamento muito grande. Porém esta estratégia não será testada, portanto não saberemos se existe algum ganho significativo com esta abordagem.

5.1.1.1 Serviço Implementado

O serviço utilizado para realizarmos os testes de desempenho consiste em um serviço baseado no primeiro cenário de utilização - Integração de um Sistema de Controle de Redes Industriais. Logo, quando um cliente requer um dos serviços disponíveis (mensagens de controle de alto nível, conforme a figura 5.4, implementadas pelos métodos `ns__start`, `ns__stop` e `ns__status`), uma mensagem SOAP é enviada à plataforma SHIP, que executa a operação e retorna uma mensagem SOAP que contém o resultado.

```
<?xml version="1.0"encoding="UTF-8"?>
<definitions name="chao"
targetNamespace="http://localhost:80/chao.wsdl"
xmlns:tns="http://localhost:80/chao.wsdl"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="urn:chao"
xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/”>

<types>
<documentation>
</documentation>
<schema targetNamespace="urn:chao"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="urn:chao"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified”>
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
</schema>
</types>

<message name="startRequest”></message>
<message name="startResponse”><part name="statusResult"type="xsd:int"/></message>
<message name="stopRequest”></message>
<message name="stopResponse”><part name="statusResult"type="xsd:int"/></message>
<message name="statusRequest”></message>
<message name="statusResponse”><part name="statusResult"type="xsd:int"/></message>

<portType name="chaoPortType”>
<operation name="start”>
<documentation>Service definition of function ns__start</documentation>
<input message="tns:startRequest"/>
<output message="tns:startResponse"/>
</operation>
<operation name="stop”>
<documentation>Service definition of function ns__stop</documentation>
<input message="tns:stopRequest"/>
<output message="tns:stopResponse"/>
</operation>
<operation name="status”>
<documentation>Service definition of function ns__status</documentation>
<input message="tns:statusRequest"/>
<output message="tns:statusResponse"/>
</operation>
</portType>

<binding name="chao"type="tns:chaoPortType”>
<SOAP:binding style="rpc"transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="start”>
<SOAP:operation style="rpc"soapAction=/>
<input>
<SOAP:body use="encoded"namespace="urn:chao"encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<SOAP:body use="encoded"namespace="urn:chao"encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="stop”>
<SOAP:operation style="rpc"soapAction=/>
<input>
<SOAP:body use="encoded"namespace="urn:chao"encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<SOAP:body use="encoded"namespace="urn:chao"encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="status”>
<SOAP:operation style="rpc"soapAction=/>
<input>
<SOAP:body use="encoded"namespace="urn:chao"encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<SOAP:body use="encoded"namespace="urn:chao"encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>

<service name="chao”>
<documentation>gSOAP 2.7.3 generated service definition</documentation>
<port name="chao"binding="tns:chao”>
<SOAP:address location="http://localhost:80"/>
</port>
</service>
</definitions>
```

Figura 5.5: WSDL gerado a partir da definição dos serviços e tipos dos dados

Após este processamento, o servidor começa a esperar por outras chamadas aos serviços. Convém lembrar que para esse caso de uso não utilizamos a nossa abordagem de classificação de serviços.

5.1.2 Utilização de Sistemas Embutidos no Monitoramento e Controle de Pacientes

Um outro exemplo de uso da nossa proposta é a utilização de sistemas embutidos no monitoramento e controle de pacientes, de acordo com a figura 5.6. Uma vez que os sinais vitais destes estejam sendo constantemente monitorados/controlados por equipamentos, que na nossa abordagem seriam os dispositivos remotos ligados às plataformas, estas fariam a aquisição dos sinais vitais (seguindo parâmetros pré-programados, isto é, aqueles que possuírem requisitos de tempo real têm maior prioridade), e disponibilizariam via XML/SOAP as informações para o servidor de aplicação.

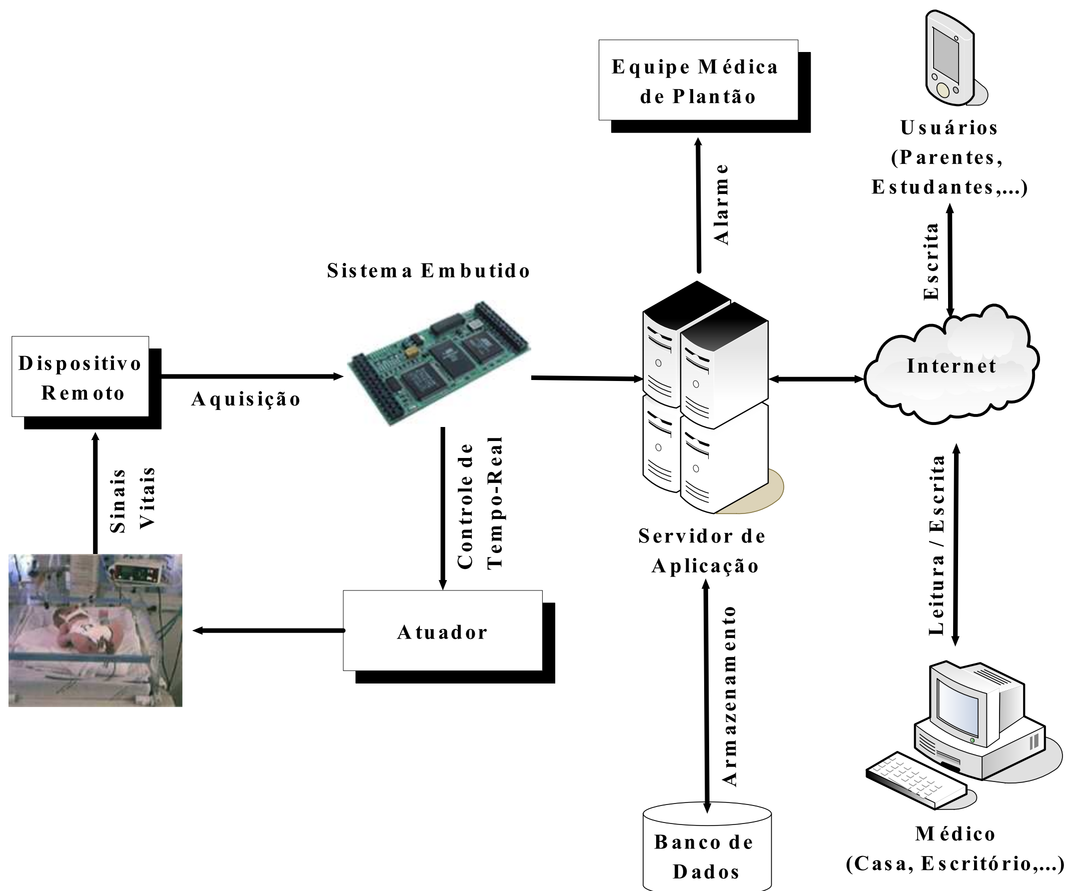


Figura 5.6: Cenário de Utilização para Monitoramento e Controle de Pacientes

O papel do servidor de aplicação seria o de distribuir as diversas informações provindas de várias configurações como essa, dentro do ambiente hospitalar, para os respectivos clientes (a base de dados do hospital e/ou médicos em casa, no consultório ou no próprio hospital, utilizando PCs, Palms, etc).

Neste exemplo podemos ver claramente a diferenciação dos serviços: *gold* para o controle em tempo real, *silver* para os alarmes enviados à equipe médica de plantão, *bronze* para a atualização da base de dados e *best effort* para os demais serviços. A figura 5.7 apresenta, as definições (em C) dos tipos dos dados utilizados pelos métodos descritos acima.

```
//gsoap ns service name:  hospital
//gsoap ns service name:  hospital
//gsoap ns service style:  rpc
//gsoap ns service encoding:  encoded
//gsoap ns schema namespace:  urn:hospital
int ns__controle(int *result);
int ns__alarme(int *result);
int ns__BD(int *result);
int ns__outros(int *result);
```

Figura 5.7: Definições em C

5.1.2.1 Serviço Implementado

Os testes para a verificação do classificador de serviços consistem em um serviço baseado no segundo cenário de utilização - Utilização de Sistemas Embutidos no Monitoramento e Controle de Pacientes. Portanto temos 4 classes de serviços, cada uma referente a um tipo de método. Logo, quando um cliente requer um dos serviços disponíveis (conforme a figura 5.7, implementadas pelos métodos `ns__controle`, `ns__alarme`, `ns__BD` e `ns__outros`), uma mensagem SOAP é enviada à plataforma SHIP, que executa a operação e retorna uma mensagem SOAP que contém o resultado. Após isso, o servidor começa a esperar por outras chamadas aos serviços.

5.2 Avaliação do Ambiente de Execução

Esses dois testes buscam apenas avaliar se é viável trabalhar com *Web Services* em sistemas embutidos a fim de estimar a sua capacidade de processar requisições em situações de sobrecarga e levando em conta a prioridade dos serviços requisitados.

Para geração de testes e análise das requisições SOAP/XML-RPC enviadas aos *web services* disponibilizados na plataforma SHIP via gSOAP usamos o software JMeter 2.1.1 (Jakarta, 2005) rodando em um Pentium 4 2.8GHz com Windows XP e Java 1.4.2.

5.2.1 JMeter

Para obtenção do resultados configuramos o software, estabelecendo o número de *threads*, o período de *ramp-up* e a quantidade de passadas (*loops*) que o teste será realizado, conforme figura 5.8.

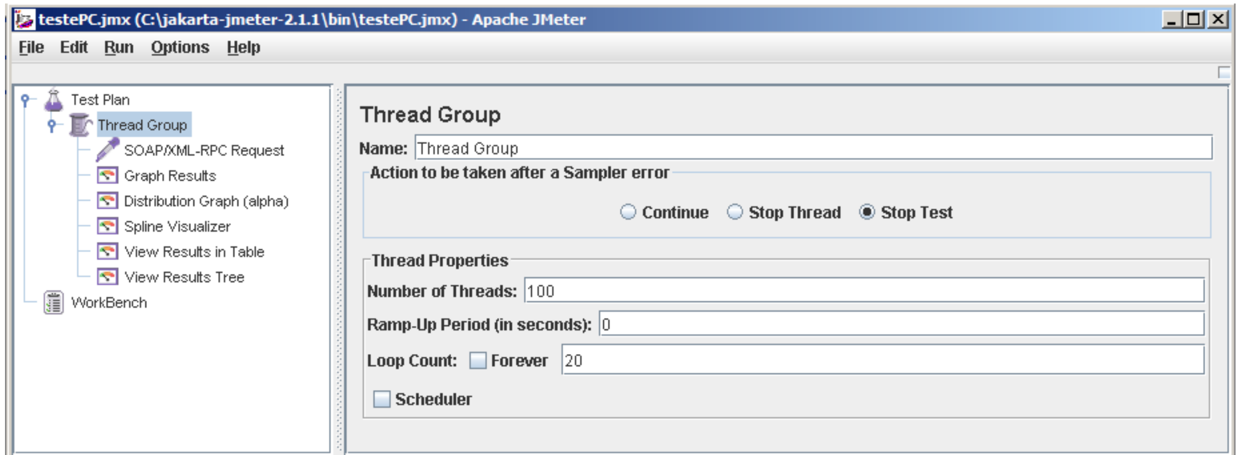


Figura 5.8: Configuração do JMeter

O período de *ramp-up* estabelece a quantidade de tempo que o JMeter utilizará para a criação do número total de clientes. Com o valor 0 o JMeter irá criar todos os clientes imediatamente. Se o período de *ramp-up* for definido para T segundos (neste caso 0, 1, 2), e o número total de clientes é N (neste exemplo, 2), o JMeter irá criar um cliente a cada T/N segundos.

Os valores estatísticos disponibilizados pelo JMeter que utilizamos são a média do tempo de resposta (latência) e a vazão (*troughput*) - mensurada pela relação número de requisições por

minuto.

5.2.2 Testes de Desempenho

Como podemos observar nos resultados dos testes de desempenho, referente ao web service implementado na subseção 5.1.1.1, apresentados na figura 5.9, obtidos ao executarmos o método `ns__start`, quando um, dois ou quatro clientes realizam requisições aos serviços sem que haja concorrência entre essas requisições, ou seja, sem a retransmissão de pacotes, a média do tempo de resposta (latência) e o desvio padrão são basicamente os mesmos - 24 milisegundos e 8, respectivamente, além de obtermos os mesmos valores para o tempo de resposta mínimo e máximo, bem como a mediana. Nos testes realizados, todas as requisições SOAP/XML-RPC, por invocarem um único método, possuem 481 bytes.

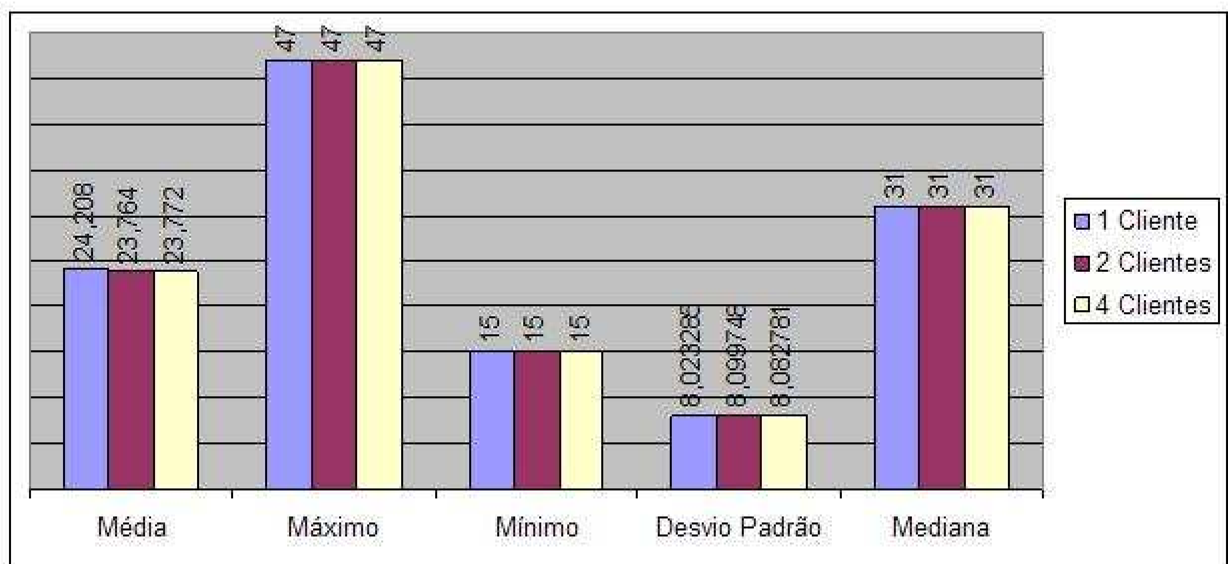
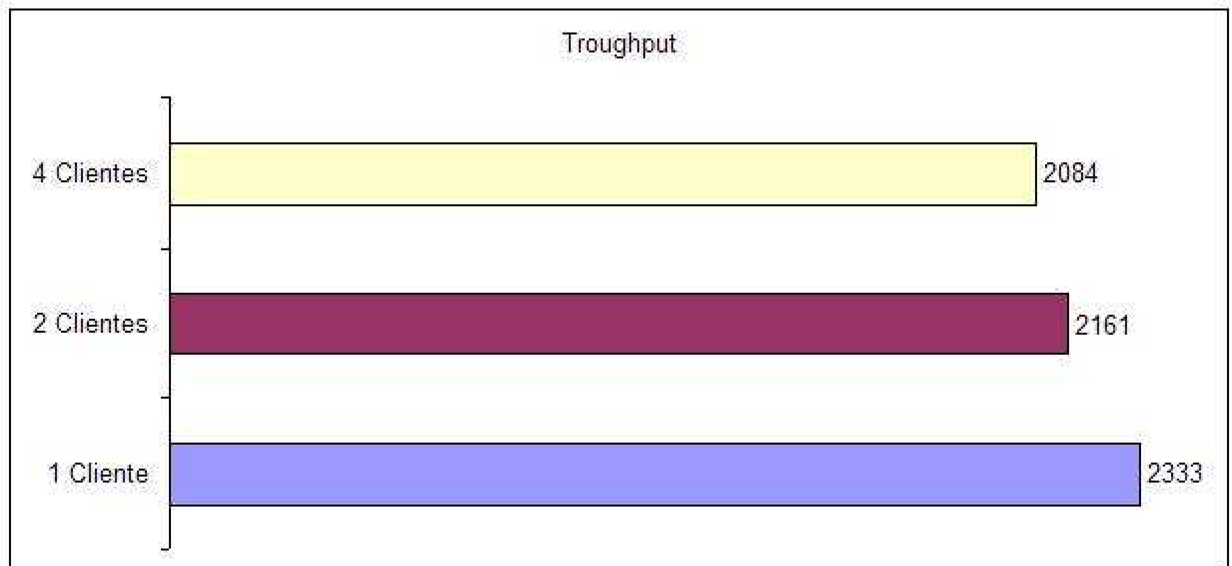
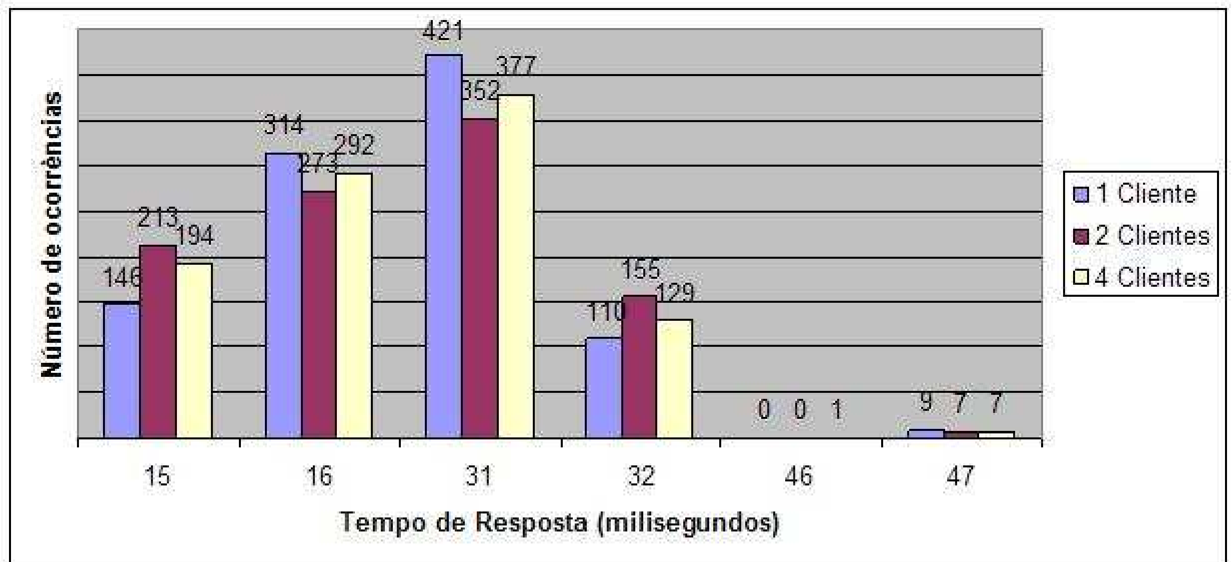


Figura 5.9: Tempo de Resposta - 1, 2 e 4 clientes sem concorrência

Outra informação importante é que com um cliente temos uma melhor vazão (*troughput*) conforme figura 5.10 A - mensurada pela relação número de requisições por minuto. Por fim, a figura 5.10 B ilustra o número de ocorrências de cada tempo de resposta obtido por cada cliente em 1000 requisições, que mostram que ou a fila está vazia e que a plataforma atende direto a requisição (15-16 ms) ou que esta tem que atender 1 ou 2 requisições que estão na fila antes (31-32 e 46-47 ms, respectivamente).



(A) Vazão - requisições por minuto



(B)

Figura 5.10: Vazão e Número de Ocorrências - 1, 2 e 4 clientes sem concorrência

5.2.3 Testes do Classificador

Os resultados dos testes do classificador, referente ao *web service* implementado na subseção 5.1.2.1, apresentados na figura 5.11, e apresentados em escala logarítmica, foram obtidos ao executarmos os métodos `ns__controle`, `ns__alarme`, `ns__BD` e `ns__outros`, através da invocação de quatro clientes (um para cada método) que chamam requisições aos serviços ao mesmo tempo, gerando assim, concorrência entre essas requisições.

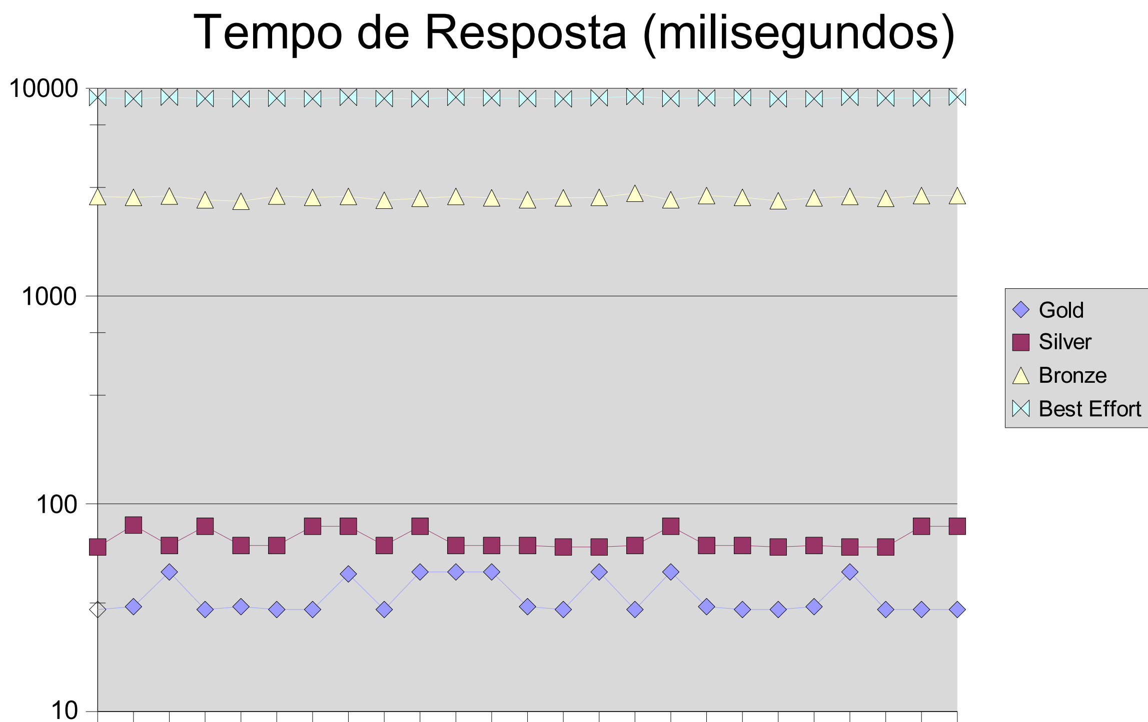


Figura 5.11: 4 clientes de classes diferentes e com concorrência

Como podemos observar os resultados demonstram que o classificador funciona corretamente provendo uma diferenciação de serviços absoluta (fim-a-fim), priorizando as requisições cujo o método, por definição do desenvolvedor, é classificado como gold e portanto realizando o enfileiramento e processamento deste antes dos outros métodos, e de forma sucessiva realiza o processamentos dos métodos restantes (silver, bronze e best-effort).

Nos testes realizados, as requisições SOAP/XML-RPC possuem, respectivamente, 477, 473, 465 e 473 bytes e seguiam a seguinte distribuição de carga:

- Criação e despacho simultâneo, via jMeter, de 4 métodos com classes diferentes;

- Aguardo da execução de todos os métodos;
- Repetição dos passos anteriores por 25 vezes.

5.2.3.1 Impacto do classificador no desempenho

Criamos um método que efetivamente não realizava nenhuma ação para analisarmos o impacto que o classificador efetuava no desempenho do ambiente de execução. Assim, num primeiro momento chamamos este método sem que utilizássemos a extensão do gSOAP, obtendo um tempo médio de resposta na ordem de 24ms.

Em outro momento, utilizando a nossa extensão do gSOAP, definimos que este método seria de maior prioridade, ou seja, um método *gold*, e ao compararmos os valores médios do tempo de resposta deste método (36ms) com os apresentados acima, podemos estimar em aproximadamente 12ms o tempo de sobrecarga (criar, inserir e remover um método de uma lista) que o classificador impõe.

5.3 Considerações Finais

Através da implementação dos serviços descritos nesse capítulo conseguimos demonstrar que podemos disponibilizar uma plataforma para o desenvolvimento de *Web Services* em sistemas embutidos cuja avaliação, tanto em relação ao desempenho, quanto ao classificador de serviços proposto, é plenamente possível.

Os resultados apresentados se mostraram bastante promissores, uma vez que o *overhead* de informação - caracterizado pelo *parsing* dos dados para um formato texto (XML), que será envelopado por um protocolo para comunicação (SOAP) - imposto por um *Web Service* (GOVINDARAJU et al., 2004), é um fator crucial de desempenho, principalmente neste tipo de ambiente onde as restrições funcionais são mais severas.

Os resultados dos testes, demonstraram que nossa abordagem é adequada. Convém lembrar que todos os aspectos de classificação dos serviços ficaram a cargo do servidor. Ou seja, no

momento que desenvolvemos o *Web Service* (*service provider*) é que definimos a qual classe pertence cada método.

6 Conclusões

Web Services apresentam-se como uma forma de interconexão de aplicações, através da Internet, entre sistemas computacionais. Além disso, por possuírem uma arquitetura eminentemente aberta e padronizada, os *Web Services* têm um grande potencial de uso para a computação distribuída.

A possibilidade dessa tecnologia ser apenas um modismo ou realmente algo que irá se tornar uma realidade comercial, discutida em (VAUGHAN-NICHOLS, 2002), já foi superada, visto que praticamente todas as soluções apresentadas no mercado atual e que têm base a *Web*, possuem, se não por completo, em partes, alguma arquitetura de desenvolvimento que envolve *Web Services*.

Portanto, nada mais natural em se propor a construção de um *web service* em um sistema embutido, podendo assim promover sua integração em um ambiente distribuído. E, a partir desse trabalho, esperamos possibilitar uma nova área de aplicação dos *Web Services* como meio de integração de Sistemas Embutidos a outros sistemas. Além disso, este trabalho se propõe a adicionar suporte a QoS - particularmente em relação aos requisitos temporais - em *Web Services*, visto que requisitos de QoS e sua política de utilização ainda não estão bem consolidados nesta tecnologia.

Sabendo que a integração de dispositivos embutidos através de *Web Services* é possível, concluímos que a adequação dos *Web Services* para integração dos inúmeros equipamentos encontrados nos níveis internos de uma corporação com o intuito de fazer com que estes sejam mais facilmente reconfiguráveis e capazes de se adaptar a mudanças no ambiente de produção pode ser feita.

Como exemplo prático podemos citar os equipamentos disponíveis em uma célula de manufatura. A utilização do paradigma *Web Services* para este caso nos traz várias vantagens, tais como:

- Células de manufaturas reconfiguráveis e modulares. Ao invés de desenvolver ou comprar novos e diferentes sistemas para supervisionamento do controle e aquisição dos dados para cada rede industrial, um elemento de integração funcionará como um *middleware* para interconexão de cada célula de manufatura com todo o ambiente empresarial;
- Fácil adaptação a novos dispositivos, fluxos de controle, produtos e processos;
- Baixo custo - uma vez que utilizaremos padrões, protocolos e linguagens gratuitos e abertos, amplamente adotados na Internet, e também, neste caso, o *toolkit* de desenvolvimento, o sistema operacional e a bibliotecas de execução da plataforma embutida tem seu código aberto;
- Fácil integração entre o chão de fábrica e os outros níveis da organização.

Os resultados obtidos mostram que o desempenho é viável para um conjunto significativo de aplicações com restrições temporais do tipo melhor-esforço e de tempo real brando (*soft*). Principalmente se confrontarmos os valores obtidos com os resultados apresentados em (ELFWING; PAULSSON; LUNDBERG, 2002), onde a performance dos servidores SOAP tradicionais apresentaram resultados muito maiores que uma implementação CORBA.

Atualmente, até onde sabemos, não existe nenhum outro estudo completo de viabilidade ou experimento com a mesma intenção desta dissertação, que é a integração de dispositivos embutidos através de *Web Services*, ou seja, não encontramos nenhum caso de utilização dos produtos existentes em cenários reais provendo, assim, plataformas para disponibilização de serviços web e dispositivos de acesso à rede com baixo custo. Portanto nossa proposta contribui nesse ponto, possibilitando uma visão a mais na área e com uma maior riqueza de detalhes.

O grande mérito desta proposta é que, ao utilizar *Web Services*, não precisamos nos preocupar com o resto do ambiente distribuído, isto é, quais são os sistemas legados utilizados no

servidor de aplicação, como foi feita a modelagem do banco de dados, que tipo de software cliente os usuários estão utilizando.

Todas essas vantagens partem do princípio de que *Web Services* utilizam XML sobre uma arquitetura aberta com padrões de *facto*, logo, a utilização dessa estrutura pode ser aplicada nos mais diversos cenários, tais como: monitoramento, Controle e Difusão de dados em linhas de transmissão (elétricas, de televisão, etc) e seus diversos sistemas/aplicações envolvidos e ambientes de chão de fábrica com seus diversos sistemas embutidos com interfaces de comunicação específicas.

6.1 Limitações do Trabalho

Com base nos objetivos traçados para este trabalho, encontramos muitas dificuldades, quanto aos requisitos funcionais do ambiente utilizado (plataforma embutida) e o ajustes necessários para a utilização de *Web Services* neste.

Este trabalho não contemplou os seguintes ítens:

- O desenvolvimento de cenários mais complexos descritos na seção 5.1, envolvendo um número maior de clientes e servidores e realizando a integração destes com servidores de bancos de dados e de aplicações não serão implementados pelos problemas acima citados que tomaram todo o tempo do trabalho;
- Aspectos temporais relativos a implantação da política de escalonamento, apesar de serem considerados relevantes, não foram abordados claramente, ou seja, disponibilizamos o mecanismo de classificação, porém não trabalhamos com as funcionalidades de Tempo Real na aplicação;
- Não foram realizadas medições para inferirmos, em relação ao tempo de resposta de uma requisição, qual a participação efetiva entre o que foi gasto em comunicação e processamento, ou seja, o quanto representa nesse tempo a parte referente a rede LAN utilizada, ao processamento da mensagem e ao processamento do método chamado.

6.2 Perspectivas Futuras

Além do testes apresentados neste trabalho, outros testes, especialmente aqueles que tratam do uso de memória, processamento, e capacidade de transmissão seriam de grande valia, numa perspectiva de trabalhos futuros que fariam parte do escopo deste trabalho.

Outro aspecto que vislumbramos para o futuro seria o emprego de técnicas de escalonamento adptativo, já proposto em outras arquiteturas e comentado na seção 4.1. Além disso técnicas de compressão de dados, presentes no gSOAP, seriam uma boa solução para a otimização do desempenho do servidor.

Por fim, como principal perspectiva futura acreditamos que nossa abordagem é plenamente capaz de ser implantada em um sistema real, ou seja, em situações reais tais quais as descritas nos cenários de utilização.

Também é importante ressaltar que, por causa deste trabalho, foram detectadas várias "lacunas" no *firmware* do SHIP, o que levou a Boreste a realizar e/ou programar uma série de modificações e melhorias neste. Uma nova implementação da pilha TCP/IP com capacidade de múltiplas *threads* é um exemplo de destaque desses esforços futuros. Já a longo prazo existe a possibilidade de uma implementação própria de uma solução completa de hardware e software com suporte a *Web Services*.

Referências

ANDERSON, Anne H. An introduction to the web services policy language (wspl). *policy*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 189, 2004.

Apache. *Apache HTTP Server Project*. 1996. Disponível em: <http://httpd.apache.org/>. Acesso em: 10/11/2005.

ATMEL. *AVR Embedded Internet Toolkit*. 2005. Disponível em: <http://www.atmel.com/products/AVR/>. Acesso em: 28/02/2005.

BECKER, Aleksader Knabben; CLARO, Daniela Barreiro; SOBRAL, João Bosco. Web services e xml: Um novo paradigma da computação distribuída. *OD' 2001 - Objetos Distribuídos*, Novembro 2001. Disponível em: <http://www.inf.ufsc.br/˜danclaro/download/ArtigoWebServices.pdf>. Acesso em: 03/08/2004.

BENATALLAH, Boualem; SHENG, Quan Z.; DUMAS, Marlon. The self-serv environment for web services composition. *IEEE Internet Computing*, IEEE Computer Society, Los Alamitos, CA, USA, v. 07, n. 1, p. 40–48, 2003. ISSN 1089-7801.

BONIATI, Bruno B.; PADOIN, Edson Luiz. Web services como middlewares para interoperabilidade em sistemas. *Revista do CCEI - Centro de Ciências da Economia e Informática*, v. 7, n. 12, p. 17 – 24, Agosto 2003.

BOOTH, David et al. *Web Services Architecture*. fev. 2004. Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>. Acesso em: 07/11/2005.

BORESTE. *SHIP - Embedded Ethernet Board*. 2005. Disponível em: <http://www.boreste.com>. Acesso em: 28/02/2005.

BOSH, Robert. *CAN Specification Version 2.0*. Stuttgart: [s.n.], 1991. Disponível em: <www.algonet.se/˜staffann/developer/CAN.htm>. Acesso em: 12/12/2005.

BURNS, Alan; WELLINGS, Andy. *Real-Time Systems and Programming Languages, Third Edition*. [S.l.]: Addison Wesley, 2001. ISBN 0-201-72988-1.

CAMELO, Dioclécio Moreira. Web service. Setembro 2002. Disponível em: <http://www.cqgp.sp.gov.br/downloads/WebServices.pdf>. Acesso em: 31/07/2004.

CERVIERI, Alexandre; OLIVEIRA, Romulo Silva de; GEYER, Cláudio F. Resin. Uma abordagem de escalonamento adaptativo no ambienten real-time corba. *XX SBRC - Simpósio Brasileiro de Redes de Computadores*, Maio 2002.

CHAPPELL, David; JEWELL, Tyler. *Java Web Services*. First. [S.l.]: O'Reilly, 2002.

CONTI, Marco; KUMAR, Mohan; DAS, Sajal K.; SHIRAZI, Behrooz A. Quality of service issues in internet web services. *IEEE Transactions on Computers*, v. 51, n. 6, p. 593 – 594, June 2002.

CURBERA, Francisco; DUFTLER, Matthew; KHALAF, Rania; NAGY, William; MUKHI, Nirmal; WEERAWARANA, Sanjiva. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, v. 6, n. 2, p. 86 – 93, Mar. - Apr. 2002.

CYGWIN. *Cygwin Information and Installation*. 2005. Disponível em: <http://www.cygwin.com>. Acesso em: 22/03/2005.

DIAS, Kelvin Lopes; SADOK, Djamel Fauzi Hadj. Internet móvel: Tecnologias, aplicações e QoS. *19º Simpósio Brasileiro de Redes de Computadores*, v. 1, p. 137 – 185, Maio 2001.

ELFWING, Robert; PAULSSON, Ulf; LUNDBERG, Lars. Performance of soap in web service environment compared to corba. In: *APSEC '02: Proceedings of the Ninth Asia-Pacific Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2002. p. 84. ISBN 0-7695-1850-8.

ENGELEN, Robert A. Van; GALLIVAN, Kyle A. The gsoap toolkit for web services and peer-to-peer computing networks. In: *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. [S.l.]: IEEE Computer Society, 2002. p. 128. ISBN 0-7695-1582-7.

ENGELEN, R. van. *Pushing the SOAP Envelope with web services for scientific computing*. 2003. Disponível em: <citeseer.ist.psu.edu/vanengelen03pushing.html>.

ENGELEN, Robert van. Code generation techniques for developing light-weight xml web services for embedded devices. In: *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*. [S.l.]: ACM Press, 2004. p. 854–861. ISBN 1-58113-812-1.

ENGELEN, Robert van; GUPTA, Gunjan; PANT, Saurabh. Developing web services for c and c++. *IEEE Internet Computing*, IEEE Computer Society, Los Alamitos, CA, USA, v. 07, n. 2, p. 53–61, 2003. ISSN 1089-7801.

ETHERNUT. *Open Source Hardware and Software Project for building Embedded Ethernet Devices*. 2005. Disponível em: <http://www.ethernut.de/>. Acesso em: 28/02/2005.

EXOR. *eSOAP - Embedded SOAP*. 2005. Disponível em: <http://www.embedding.net/eSOAP/>. Acesso em: 28/02/2005.

FARINES, Jean-Marie; FRAGA, Joni da Silva; OLIVEIRA, Rômulo Silva de. *Sistemas de Tempo Real*. [S.l.]: Escola de Computação 2000, 2000.

GENIVIA. *gSOAP - C/C++ web services and clients*. 2005. Disponível em: <http://www.genivia.com/>. Acesso em: 28/02/2005.

GOUSCOS, Dimitris; KALIKAKIS, Manolis; GEORGIADIS, Panagiotis. An approach to modeling web service qos and provision price. *wisew*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 121–130, 2003.

GOVINDARAJU, Madhusudhan; SLOMINSKI, Aleksander; CHIU, Kenneth; LIU, Pu; ENGELEN, Robert van; LEWIS, Michael J. Toward Characterizing the Performance of SOAP Toolkits. *5th IEEE/ACM International Workshop on Grid Computing (short paper)*, November 2004.

IBM. *Alphaworks Emerging Technologies Toolkit*. 2005. Disponível em: <http://www.alphaworks.ibm.com/tech/ettk>. Acesso em: 28/02/2005.

ISSARNY, V.; SACCHETTI, D.; TARTANOGLU, F.; SAILHAN, F. *Enabling Ambient Intelligence via the Web*. 2002. Disponível em: <citeseer.ist.psu.edu/issarny02enabling.html>.

Jakarta. *Apache JMeter*. 2005. Disponível em: <http://jakarta.apache.org/jmeter/>. Acesso em: 07/11/2005.

JANECEK, Jan. Efficient soap processing in embedded systems. In: *11th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS)*. [S.l.: s.n.], 2004. p. 128–135.

KILGORE, Richard A. Simulation web services with .net technologies. *Proceedings of the Winter Simulation Conference*, v. 1, p. 841 – 846, 8 - 11 Dec. 2002.

KOBJECTS.ORG. *kObjects*. 2006. Disponível em: <http://kobjects.sourceforge.net/>. Acesso em: Acessado em: 13/02/2006.

KOPETZ, Hermann. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1997. ISBN 0792398947.

KUACHAROEN, P.; SHALAN, M.; MOONEY III, V. A configurable hardware scheduler for real-time systems. In: *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*. [s.n.], 2003. p. 96–101. Disponível em: <citeseer.ist.psu.edu/kuacharoen03configurable.html>.

KXML. *kXML*. 2006. Disponível em: <http://kxml.sourceforge.net/>. Acesso em: Acessado em: 13/02/2006.

LEA, Doug; VINOSKI, Steve. Middleware for web services. *IEEE Internet Computing*, v. 7, n. 1, p. 28 – 29, Jan. - Feb. 2003.

LIGHTNER Engineering. *PicoWeb Server*. 2005. Disponível em: <http://www.picoweb.net/>. Acesso em: 28/02/2005.

LUCCA, José Eduardo De. Novas tecnologias de rede. 2003. Disponível em: <http://www.inf.ufsc.br/˜delucca/apost-pdf.zip>. Acesso em: 31/07/2004.

MACHADO, Guilherme Bertoni; SIQUEIRA, Frank; MITTMANN, Robinson; VIEIRA, Carlos Augusto Vieira e. Embedded systems integration using web services. In: *Proceedings of Fifth International Conference on Networking (ICN'06)*. Mauritius Island: IEEE Computer Society Press, 2006.

Macraigor Systems. *Macraigor Systems GNU Tools*. 2005. Disponível em: <http://www.macraigor.com/full_gnu.htm>. Acesso em: 22/03/2005.

MENASCÉ, Daniel A. QoS issues in web services. *IEEE Internet Computing*, v. 6, n. 6, p. 72 – 75, Nov. - Dec. 2002.

MENASCÉ, Daniel A. Reponse-time analysis of composite web services. *IEEE Internet Computing*, v. 8, n. 1, p. 90 – 92, Jan. - Feb. 2004.

Microsoft. *Distributed Component Object Model*. nov. 1996. Disponível em: <http://msdn.microsoft.com/library/en-us/dndcom/html/msdn_dcomtec.asp>. Acesso em: 10/11/2005.

Microsoft. *Internet Information Services*. 2005. Disponível em: <http://www.microsoft.com/WindowsServer2003/iis/default.mspx>. Acesso em: 10/11/2005.

Microsoft Corporation, The. *The Microsoft .NET platform*. 2002. `http://www.microsoft.com/net/`.

NETBURNER. *NetBurner Standard hardware*. 2005. Disponível em: <http://www.netburner.com/>. Acesso em: 28/02/2005.

OMG. *Common Object Request Broker Architecture Specification*. dez. 2005. Disponível em: <http://www.omg.org>. Acesso em: 10/11/2005.

PROFIBUS. *PROFIBUS Technology and Application - System Description*. 2002. Disponível em: <www.profibus.com>. Acesso em: 12/12/2005.

RABBIT SEMICONDUCTOR. *Rabbit Ethernet Connectivity*. 2005. Disponível em: <http://www.rabbitsemiconductor.com/products/Ethernet/>. Acesso em: 28/02/2005.

REIS, Cleiton Peres. *Embedded Systems Applications*. 2003. Disponível em: <www.linuxabordo.com.br/artigos/EmbeddedSystemsApplications.pdf>. Acesso em: 25/05/2004.

ROY, Jaideep; RAMANUJAN, Anupama. Understanding web services. *IEEE Internet Computing*, v. 3, n. 6, p. 69 – 73, Nov. - Dec. 2001.

RYU, Sook-Hyun; KIM, Jae-Young; HONG, James Won-Ki. Approaches to support differentiated quality of web service. In: *QofIS*. [S.l.: s.n.], 2001. p. 273–285.

SERRA, Antônio; GAÏTI, Dominique; BARROSO, Giovanni; RAMOS, Ronaldo; BOUDY, Jérôme. Uma plataforma distribuída com balanceamento de cargas para servidores web baseada na diferenciação de serviços. *XXXI SEMISH - Seminário Integrado de Software e Hardware*, Agosto 2004.

SHARMA, Amit; ADARKAR, Hemant; SENGUPTA, Shubhashis. Managing qos through prioritization in web services. *wisew*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 140–148, 2003.

SILVA, Juarez Bento da. *Monitoramento, aquisição e controle de sinais elétricos via Web, utilizando microcontroladores*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 2002.

SIQUEIRA, Frank. Especificação de requisitos de qualidade de serviço em sistemas abertos: a linguagem qsl. *XX Simpósio Brasileiro de Redes de Computadores (SBRC)*, 2002. Disponível em: <www.inf.ufsc.br/˜frank/papers/SBRC2002.pdf>. Acesso em: 18/05/2005.

SONDA, Gláucia C. S.; MONTEZ, Carlos. Uma proposta de implementação de diferenciação de serviços na arquitetura de web services. *I Work Comp Sul*, Maio 2004.

SUN MICROSYSTEMS. *Java Remote Method Invocation – Distributed Computing for Java*. 2003. Disponível em: <http://java.sun.com/products/jdk/rmi/>. Acesso em: 12/01/2006.

SUN Microsystems. *Java 2 platform, Micro Edition (J2ME)*. 2005. Disponível em: <http://java.sun.com/j2me/>. Acesso em: 28/02/2005.

TIAN, M.; GRAMM, A.; NAUMOWICZ, T.; RITTER, H.; SCHILLER, J. *A concept for qos integration in web services*. 2003. Disponível em: <citeseer.ist.psu.edu/tian03concept.html>.

UDDI. *Universal Description, Discovery and Integration*. 2005. Disponível em: <http://www.uddi.org/>. Acesso em: Acessado em: 13/08/2004.

UNICOI Systems. *Fusion Web*. 2005. Disponível em: <http://www.unicoi.com/>. Acesso em: 28/02/2005.

UPnP Forum. *UPnP device architecture*. 2000. Disponível em: <http://www.upnp.org>. Acesso em: 12/12/2005.

VAUGHAN-NICHOLS, Steven J. Web services: Beyond the hype. *Computer*, IEEE Computer Society, Los Alamitos, CA, USA, v. 35, n. 2, p. 18–21, 2002. ISSN 0018-9162.

VINOSKI, Steve. Integration with web services. *IEEE Internet Computing*, v. 7, n. 6, p. 75 – 77, Nov. - Dec. 2003.

VOGELS, Werner. Web services are not distributed objects. *IEEE Internet Computing*, v. 7, n. 6, p. 59 – 66, Nov. - Dec. 2003.

W3C. *Extensible Markup Language (XML)*. 2005. Disponível em: <http://www.w3.org/XML/>. Acesso em: 13/08/2004.

W3C. *Extensible Markup Language (XML) 1.0 (Third Edition)*. 2005. Disponível em: <http://www.w3.org/TR/REC-xml/>. Acesso em: 16/03/2005.

W3C. *Web Services Description Language (WSDL)*. 2005. Disponível em: <http://www.w3.org/TR/wsdl>. Acesso em: Acessado em: 13/08/2004.

W3C. *World Wide Web Consortium*. 2005. Disponível em: <http://www.w3.org/>. Acesso em: 29/04/2005.

W3C. *XML Protocol Working Group*. 2005. Disponível em: <http://www.w3.org/2000/xp/Group/>. Acesso em: Acessado em: 18/03/2005.

WBC-EUROPE. Web services on a single chip. *PIM - Project Information Manual*, 2004. Disponível em: <http://www.wbc-europe.com/>. Acesso em: 28/02/2005.

WEISER, Mark. Hot Topics: Ubiquitous Computing. *IEEE Computer*, v. 26, n. 10, p. 71–72, October 1993. Disponível em: <http://www.ubiq.com/hypertext/weiser/UbiCompHotTopics.html>.

WOLF, Wayne. *Computer as Components: principles of embedded computing system desing*. [S.l.]: Morgan Kaufmann, 2001.

YUAN, K.-J. Lin S.-T. Wise-building simple intelligence into web services. In: *IEEE/WIC International Conference on Intelligent Agent Technology, proceedings*. Halifax, Canada: [s.n.], 2003. ISBN 0-7695-1931-8.

ZENG, Liangzhao; BENATALLAH, Boualem; NGU, Anne H. H.; DUMAS, Marlon; KALAGNANAM, Jayant; CHANG, Henry. Qos-aware middleware for web services composition. *IEEE Trans. Software Eng.*, v. 30, n. 5, p. 311–327, 2004.

Anexo I - Configuração, Instalação e Utilização do gSOAP

1.1 gSOAP

Através do site do fabricante - www.genivia.com - fazer o download, no sourceforge, da versão independente de plataforma (*Platform-Independent*), para utilização no ambiente Windows, via cygwin.

Há a possibilidade de versão para linux e windows (que seria integrada a um ambiente como Visual Studio).

1.2 Cygwin

Fazer o download do cygwin (www.cygwin.com).

1.3 Configuração

Descompactar a instalação do gSOAP num diretório dentro do Cygwin, executando os seguintes comandos:

* `./configure`

* `make`

* `make install`

1.4 Utilização

Criar uma pasta relacionada a aplicação a ser desenvolvida no diretório local do usuário e copiar os arquivos `typemap.dat` e `stdsoap2.c` (para aplicações em C) ou `stdsoap2.cpp` (C++) para esta pasta, ou se preferir criar uma pasta dentro do diretório do gSOAP.

Se possuir o wsdl do *web service* para a criação da aplicação quanto do cliente usar o *parser* (utilizando a diretiva -c para a definição dos serviços e tipos dos dados em C):

* `wsdl2h.exe -c arquivo.wsdl`

Gerando a definição dos serviços e tipos dos dados através do `arquivo.h`, ou definir esse arquivo manualmente.

Compilar esse arquivo com o gSOAP Stub and Skeleton Compiler for C and C++ (utilizando a diretiva -c para a criação do ambiente de *runtime* em C):

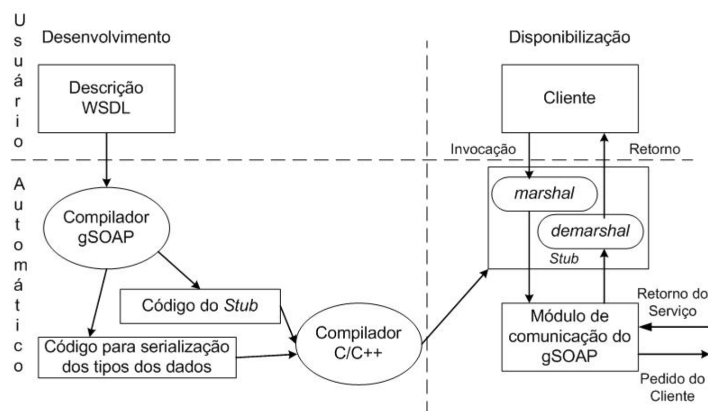
* `soapcpp2.exe -c arquivo.h`

Logo, é criado todo o ambiente de *runtime* tanto para a criação de uma aplicação quanto de um cliente.

1.4.1 Cliente

Criar o `cliente.c`

compilar: `gcc -o cliente cliente.c soapC.c soapClient.c stdsoap2.c`

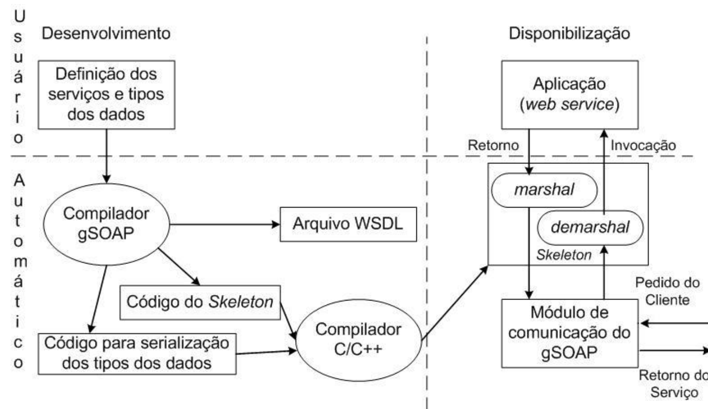


Desenvolvendo e Disponibilizando um cliente com gSOAP

1.4.2 Aplicação

Criar o `server.c`

compilar: `gcc -o server server.c soapC.c soapServer.c stdsoap2.c`



Desenvolvendo e Disponibilizando um serviço com gSOAP

Essa aplicação pode ser compilada como um cgi e disponibilizada em qualquer web server (apache, por exemplo), para isso manter o seguinte trecho de código (comentado) durante a criação da aplicação:

Procedimento 3 Método principal de uma aplicação

```
int main(int argc, char **argv)
{ int m, s; /* master and slave sockets */
 struct soap soap;
 soap_init(&soap);
/* if (argc < 2)
  soap_serve(&soap); */
// else
// {
  m = soap_bind(&soap, NULL, 80, 100);
  if (m < 0)
  { soap_print_fault(&soap, stderr);
   exit(-1);
  }
  fprintf(stderr,"Socket connection successful:  master socket = %d\n",m);
  for ( ; ; )
  { s = soap_accept(&soap);
   fprintf(stderr,"Socket connection successful:  slave socket = %d\n",s);
   if (s < 0)
   { soap_print_fault(&soap, stderr);
    exit(-1);
   }
   soap_serve(&soap);
   soap_end(&soap);
  }
// }
 return 0;
}
```

Do contrário a aplicação funcionará como um *stand-alone web server*.

Anexo II - Exemplo de Aplicação

2.1 Calculadora

Baseado na pasta de exemplos disponível na distribuição do gSOAP, é apresentado, como exemplo, o código para a criação da aplicação (*stand-alone*) e do cliente de um serviço de calculadora.

O arquivo de definição do tipo dos dados `calc.h` fica da seguinte forma:

Procedimento 4 calc.h

```
//gsoap ns service name:  calc
//gsoap ns service style:  rpc
//gsoap ns service encoding:  encoded
//gsoap ns schema namespace:  urn:calc
int ns__add(double a, double b, double *result);
int ns__sub(double a, double b, double *result);
int ns__mul(double a, double b, double *result);
int ns__div(double a, double b, double *result);
```

Obs: Se o usuário resolvesse criar o arquivo `calc.h` através de um arquivo `calc.wsdl`, o arquivo `typemap.dat` precisa estar na pasta (ou linkado através de um *makefile*) para que este possa utilizar o *parser* `wsdl2h.exe`. Outro detalhe, é que caso o usuário não tenha o cygwin instalado, a biblioteca deste (`cygwin1.dll`) deve ser anexada na distribuição da aplicação e/ou do cliente.

2.1.1 Cliente

No cliente devemos definir se a aplicação a ser requisitada esta disponível em um *web service* ou é uma aplicação *stand-alone*, conforme linha comentada no código.

2.1.2 Aplicação

Na aplicação também partimos do mesmo princípio, neste exemplo a aplicação é do tipo *stand-alone*.

Procedimento 5 calcclient.c

```
#include "soapH.h"
#include "calc.nsmap"

//const char server[] = "http://endereco/calcserver.cgi";
const char server[] = "localhost";

int main(int argc, char **argv)
{ struct soap soap;
 double a, b, result;
 if (argc < 4)
 { fprintf(stderr, "Usage:  [add|sub|mul|div] num num\n");
  exit(0);
 }
 soap_init(&soap);
 a = strtod(argv[2], NULL);
 b = strtod(argv[3], NULL);
 switch (*argv[1])
 { case 'a':
   soap_call_ns__add(&soap, server, "", a, b, &result);
   break;
  case 's':
   soap_call_ns__sub(&soap, server, "", a, b, &result);
   break;
  case 'm':
   soap_call_ns__mul(&soap, server, "", a, b, &result);
   break;
  case 'd':
   soap_call_ns__div(&soap, server, "", a, b, &result);
   break;
  default:
   fprintf(stderr, "Unknown command\n");
   exit(0);
 }
 if (soap.error)
 { soap_print_fault(&soap, stderr);
  exit(1);
 }
 else
  printf("result = %g\n", result);
  soap_destroy(&soap);
  soap_end(&soap);
  soap_done(&soap);
  return 0;
}
```

Procedimento 6 calcserver.c

```
#include "soapH.h"
#include "calc.nsmap"

int main(int argc, char **argv)
{ int m, s; /* master and slave sockets */
 struct soap soap;
 soap_init(&soap);
/*if (argc < 2)
 soap_serve(&soap); */
//else
//{
 m = soap_bind(&soap, NULL, 80, 100);
 if (m < 0)
 { soap_print_fault(&soap, stderr);
  exit(-1);
 }
 fprintf(stderr, "Socket connection successful:  master socket = %d\n", m);
 for ( ; ; )
 { s = soap_accept(&soap);
  fprintf(stderr, "Socket connection successful:  slave socket = %d\n", s);
  if (s < 0)
  { soap_print_fault(&soap, stderr);
   exit(-1);
  }
  soap_serve(&soap);
  soap_end(&soap);
 }
//}
return 0;
}

int ns__add(struct soap *soap, double a, double b, double *result)
{ *result = a + b;
 return SOAP_OK;
}

int ns__sub(struct soap *soap, double a, double b, double *result)
{ *result = a - b;
 return SOAP_OK;
}

int ns__mul(struct soap *soap, double a, double b, double *result)
{ *result = a * b;
 return SOAP_OK;
}

int ns__div(struct soap *soap, double a, double b, double *result)
{ *result = a / b;
 return SOAP_OK;
}
```

Anexo III - Arquivos de configuração do gSOAP modificados

3.1 Arquivo stdsoap2.h.patch

Procedimento 7 stdsoap2.h.patch

```
446,448c446
< #ifndef WITH_NOIO
< # ifndef WIN32
< # ifndef PALM
---
> #if !defined(WITH_NOIO) && !defined(WIN32) && !defined(PALM)
453,454c451
< # ifndef VXWORKS
< # ifndef SYMBIAN
---
> # if defined(HAVE_STRINGS_H) && !defined(VXWORKS) && !defined(SYMBIAN)
456d452
< # endif
474,475d469
< # endif
< # endif
512c506,507
< # include <math.h> /* for isnan() */
---
> //# include <math.h> /* for isnan() */
> # define isnan(n) 0
550a546,557
> /* Boreste:  portability issues.
> Values conforming to glibc-2.2+.  */
> #ifndef SHUT_RD
> #define SHUT_RD 0
> #endif
> #ifndef SHUT_WR
> #define SHUT_WR 1
> #endif
> #ifndef SHUT_RDWR
> #define SHUT_RDWR 2
> #endif
>
```

3.2 Arquivo soapdefs.h

Procedimento 8 soapdefs.h

```
/* $Id:  soapdefs.h,v 1.1 2005/08/08 18:15:35 carlos Exp $
 *
 * File:  soapdefs.h
 * Module:  gsoap
 * Project:  AT91X40DK
 * Author:  Carlos Augusto Vieira e Vieira <carlos.vieira@boreste.com>
 * Target:
 * Comment:
 * Copyright(c) 2005 Boreste (CNX Technologies).  All Rights Reserved.
 *
 */

#ifndef HAVE_CONFIG_H
#define HAVE_CONFIG_H
#endif

#ifndef WITH_LEANER
#define WITH_LEANER
#endif

//#ifndef WITH_NOHTTP
//#define WITH_NOHTTP
//#endif

#ifndef WITH_NOIDREF
#define WITH_NOIDREF
#endif

#define isnan(n) 0

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

3.3 Arquivo config.h

Procedimento 9 config.h

```
/* config.h.  Generated by configure.  */
/* config.h.in.  Generated from configure.in by autoheader.  */

/* Define to 1 if you have the `alarm' function.  */
//#define HAVE_ALARM 0

/* Define to 1 if you have the <arpa/inet.h> header file.  */
#define HAVE_ARPA_INET_H 1

/* Define to 1 if you have the <errno.h> header file.  */
#define HAVE_ERRNO_H 1

/* Define to 1 if you have the <fcntl.h> header file.  */
#define HAVE_FCNTL_H 1

/* Define to 1 if you have the `ftime' function.  */
//#define HAVE_FTIME 0

/* Define to 1 if you have the `gethostbyname_r' function.  */
//#define HAVE_GETHOSTBYNAME_R 1

/* Define to 1 if you have the `gettimeofday' function.  */
//#define HAVE_GETTIMEOFDAY 0

/* Define to 1 if you have the `gmtime' function.  */
//#define HAVE_GMTIME 0

/* Define to 1 if you have the `gmtime_r' function.  */
#define HAVE_GMTIME_R 1

/* Define to 1 if you have the <inttypes.h> header file.  */
//#define HAVE_INTTYPES_H 0

/* Define to 1 if you have the <limits.h> header file.  */
#define HAVE_LIMITS_H 1
```

Procedimento 10 config.h - continuação

```
/* Define to 1 if you have the `localtime_r' function. */
//#define HAVE_LOCALTIME_R 0

/* Define to 1 if your system has a GNU libc compatible `malloc' function,
and to 0 otherwise. */
//#define HAVE_MALLOC 0

/* Define to 1 if you have the `mbtowc' function. */
//#define HAVE_MBTOWC 0

/* Define to 1 if you have the <memory.h> header file. */
//#define HAVE_MEMORY_H 0

/* Define to 1 if you have the `memset' function. */
#define HAVE_MEMSET 1

/* Define to 1 if you have the <netdb.h> header file. */
#define HAVE_NETDB_H 1

/* Define to 1 if you have the <netinet/in.h> header file. */
#define HAVE_NETINET_IN_H 1

/* Define to 1 if you have the `rand_r' function. */
//#define HAVE_RAND_R 0

/* Define to 1 if you have the `select' function. */
#define HAVE_SELECT 1

/* Define to 1 if you have the `socket' function. */
#define HAVE_SOCKET 1

/* Define to 1 if you have the `sscanf' function. */
//#define HAVE_SSCANF 0

/* Define to 1 if you have the <stdint.h> header file. */
#define HAVE_STDINT_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#define HAVE_STDLIB_H 1

/* Define to 1 if you have the `strchr' function. */
#define HAVE_STRCHR 1

/* Define to 1 if you have the `strerror' function. */
//#define HAVE_STRERROR 0

/* Define to 1 if you have the `strftime' function. */
//#define HAVE_STRFTIME 0

/* Define to 1 if you have the <strings.h> header file. */
//#define HAVE_STRINGS_H 1

/* Define to 1 if you have the <string.h> header file. */
#define HAVE_STRING_H 1

/* Define to 1 if you have the `strrchr' function. */
#define HAVE_STRRCHR 1
```

Procedimento 11 config.h - continuação

```
/* Define to 1 if you have the `strstr' function.  */
#define HAVE_STRSTR 1

/* Define to 1 if you have the `strtod' function.  */
//#define HAVE_STRTOD 0

/* Define to 1 if you have the `strtof' function.  */
//#define HAVE_STRTOF 0

/* Define to 1 if you have the `strtol' function.  */
#define HAVE_STRTOL 1

/* Define to 1 if you have the `strtoll' function.  */
//#define HAVE_STRTOLL 0

/* Define to 1 if you have the `strtoul' function.  */
#define HAVE_STRTOUL 1

/* Define to 1 if you have the `strtoull' function.  */
//#define HAVE_STRTOULL 0

/* Define to 1 if you have the <sys/socket.h> header file.  */
#define HAVE_SYS_SOCKET_H 1

/* Define to 1 if you have the <sys/stat.h> header file.  */
#define HAVE_SYS_STAT_H 1

/* Define to 1 if you have the <sys/timeb.h> header file.  */
//#define HAVE_SYS_TIMEB_H 0

/* Define to 1 if you have the <sys/time.h> header file.  */
#define HAVE_SYS_TIME_H 1

/* Define to 1 if you have the <sys/types.h> header file.  */
#define HAVE_SYS_TYPES_H 1

/* Define to 1 if you have the `timegm' function.  */
//#define HAVE_TIMEGM 0

/* Define to 1 if you have the <unistd.h> header file.  */
#define HAVE_UNISTD_H 1

/* Define to 1 if you have the `wctomb' function.  */
//#define HAVE_WCTOMB 0

/* Name of package */ #define
PACKAGE "soapcpp2"

/* Define to the address where bug reports for this package should be sent.
*/
#define PACKAGE_BUGREPORT ""

/* Define to the full name of this package.  */
#define PACKAGE_NAME "soapcpp2"
```

Procedimento 12 config.h - continuação

```
/* Define to the full name and version of this package.  */
#define PACKAGE_STRING "soapcpp2 2.7"

/* Define to the one symbol short name of this package.  */
#define PACKAGE_TARNAME "soapcpp2"

/* Define to the version of this package.  */
#define PACKAGE_VERSION "2.7"

/* Define as the return type of signal handlers (‘int’ or ‘void’).  */
#define RETSIGTYPE void

/* Define to 1 if you have the ANSI C header files.  */
#define STDC_HEADERS 1

/* Define to 1 if you can safely include both <sys/time.h> and <time.h>.
*/
#define TIME_WITH_SYS_TIME 1

/* Define to 1 if your <sys/time.h> declares ‘struct tm’.  */
/* #undef TM_IN_SYS_TIME */

/* Version number of package */
#define VERSION "2.7"

/* Define to 1 if ‘lex’ declares ‘yytext’ as a ‘char *’ by default, not a
‘char[]’.  */
#define YYTEXT_POINTER 1

/* Define to empty if ‘const’ does not conform to ANSI C. */
/* #undef const */

/* Define to rpl_malloc if the replacement function should be used.  */
/* #undef malloc */

/* Define to ‘unsigned’ if <sys/types.h> does not define.  */
/* #undef size_t */
```

Anexo IV - Makefiles

4.1 Makefile - compilação do Web Service

Procedimento 13 makefile - Compilação do Web Service

```
# $Id:  Makefile,v 1.1.2.1 2006/03/31 01:43:45 devel1 Exp $
#
# File:  Makefile
# Module:  gSOAP for DevKit
# Project:
# Author:  Carlos Augusto Vieira e Vieira (carlos.vieira@boreste.com)
# Target:  256KB RAM DevKit
# Comment:
# Copyright(c) 2003-2005 CNX Technologies.  All Rights Reserved.

PROJECT = testsoap
CLIENT = calc

SOAPRUNTIMEPATH = soap_runtime
SDKPATH = ../../../..
SOAPPATH = $(SDKPATH)/ports/gsoap

LIBS = gsoap nops tcpip_lg mbuf_lg mlink_lg c a7e10x2 gcc

PREFIX = arm-elf

SERVER_SOAPOBJ = $(SOAPRUNTIMEPATH)/soapServer.$(PREFIX).o \
      $(SOAPRUNTIMEPATH)/soapC.$(PREFIX).o
#CLIENT_SOAPOBJ = $(SOAPRUNTIMEPATH)/soapClientLib.o

HFILES = $(wildcard ./*.h) $(wildcard ./include/*.h)
SERVER_CFILES = $(wildcard ./*.c)
SFILES = $(wildcard ./*.S) # Assembly always belongs to server.
SERVER_OFILES = $(SERVER_CFILES:.c=.$(PREFIX).o) $(SFILES:.S=.$(PREFIX).o)

INCPATH = $(SDKPATH)/include $(SOAPPATH)/include $(SOAPRUNTIMEPATH)
./include
LIBPATH = $(SDKPATH)/lib
TOOLS = $(SDKPATH)/tools

CDEFS= _SYSCALLS_ ENABLE_STAT WITH_SOAPDEFS_H
#DEBUG

ROM = 0x01010000
RAM = 0x00000100
CPU = arm7tdmi
```

Procedimento 14 makefile - Compilação do Web Service - continuação

```
DBGOPTS = -g
THUMBOPTS = -mcpu=$(CPU) -mthumb -mthumb-interwork -mno-tpcs-frame
#-mtpcs-frame
#-mno-tpcs-frame
OPTIONS = $(THUMBOPTS) $(DBGOPTS)

SFLAGS = -Wall
CFLAGS = -Wall -O -mno-sched-prolog $(addprefix -D,$(CDEFS))
LDFLAGS = -nostdlib -Ttext $(ROM) -Tdata $(RAM) $(addprefix -L,$(LIBPATH))
CRT0 = $(SDKPATH)/lib/crt0.o
CC = $(PREFIX)-gcc
LD = $(PREFIX)-gcc
AS = $(PREFIX)-gcc
AR = $(PREFIX)-ar
OBJCOPY = $(PREFIX)-objcopy
OBJDUMP = $(PREFIX)-objdump
STRIP = $(PREFIX)-strip

all:  Makefile $(PROJECT).bin $(PROJECT).sym
#$(PROJECT).lst

runtime:
	@cd $(SOAPRUNTIMEPATH) && $(MAKE)

runtime-clean:
	@cd $(SOAPRUNTIMEPATH) && $(MAKE) clean

clean:  runtime-clean
	@rm -fv $(SERVER_OFILES) $(PROJECT).lst $(PROJECT).sym $(PROJECT).elf \
	$(PROJECT).bin $(PROJECT).code $(PROJECT).data

load:  all
	$(TOOLS)/upload.sh $(PROJECT).bin

jtagload:  all
	$(TOOLS)/tftp_load.sh -a $(ROM) -r $(PROJECT).bin

$(PROJECT).elf:  Makefile $(SERVER_OFILES)
	$(LD) $(OPTIONS) $(LDFLAGS) $(CRT0) $(SERVER_OFILES) $(SERVER_SOAPOBJ) \
	$(addprefix -l,$(LIBS)) -o $@

$(PROJECT).bin:  Makefile $(PROJECT).elf
	$(OBJCOPY) -R .data -O binary $(PROJECT).elf $(PROJECT).code
	$(OBJCOPY) -j .data -O binary $(PROJECT).elf $(PROJECT).data
	cat $(PROJECT).code $(PROJECT).data > $@

$(PROJECT).lst:  Makefile $(PROJECT).elf
	$(OBJDUMP) -w -D -t -S -r -z $(PROJECT).elf | sed '/^[0-9,a-f]\{8\} .[
]*d[f]\?.*$$/d' > $@

$(PROJECT).sym:  Makefile $(PROJECT).elf
	$(OBJDUMP) -t $(PROJECT).elf | sed '/^[0-9,a-f]\{8\} .[
]*d[f]\?.*$$/d;/^SYMBOL.*$$/d;/^.*file format.*$$/d;/^$$/d' | sort > $@

%.$(PREFIX).o :  %.c Makefile $(HFILES) runtime
	$(CC) $(OPTIONS) $(CFLAGS) $(addprefix -I,$(INCPATH)) -o $@ -c $<

%.$(PREFIX).o :  %.S Makefile runtime
	$(AS) $(OPTIONS) $(SFLAGS) $(addprefix -I,$(INCPATH)) -o $@ -c $<
```

4.2 Makefile - Criação do ambiente de Execução

Procedimento 15 Criação do ambiente de Execução

```
# $Id:  Makefile,v 1.1.2.1 2006/03/31 01:43:45 devel1 Exp $
#
# File:  Makefile
# Module:  gSOAP for DevKit
# Project:
# Author:  Carlos Augusto Vieira e Vieira (carlos.vieira@boreste.com)
# Target:  256KB RAM DevKit
# Comment:
# Copyright(c) 2003-2005 CNX Technologies.  All Rights Reserved.

# TODO: generate everything from WSDL file.

# This is the service definition as created by wsdl2h
SOAPSERVICE = calc.wsdl
SOAPTYPEMAP = test.typemap
SOAPSERVICE_H = $(SOAPSERVICE:.wsdl=.h)

SDKPATH = ../../../../../../devkit
SOAPPATH = $(SDKPATH)/ports/gsoap

PREFIX = arm-elf

CONST_SOAPCLEAN = soapH.h soapStub.h soapC.c soapClient.c soapServer.c \
         soapClientLib.c soapServerLib.c
SOAP_CLEANFILES = $(CONST_SOAPCLEAN) \
         $(SOAPSERVICE_H:.h=.nsmap) $(SOAPSERVICE_H) \ $(wildcard ./*.xml)
$(wildcard ./*.xsd)

CLIENT_CFILE = soapClientLib.c
CLIENT_OFILE = $(CLIENT_CFILE:.c=.o)

SERVER_CFILE1 = soapServer.c
SERVER_OFILE1 = $(SERVER_CFILE1:.c=.$(PREFIX).o)
SERVER_CFILE2 = soapC.c
SERVER_OFILE2 = $(SERVER_CFILE2:.c=.$(PREFIX).o)
SERVER_OFILES = $(SERVER_OFILE1) $(SERVER_OFILE2)

INCPATH = $(SDKPATH)/include $(SOAPPATH)/include
LIBPATH = $(SDKPATH)/lib

CDEFS= _SYSCALLS_ ENABLE_STAT WITH_SOAPDEFS_H
#WITH_NOSERVEREQUEST
```

Procedimento 16 Criação do ambiente de Execução - continuação

```
ROM = 0x01010000
RAM = 0x00000100
CPU = arm7tdmi

DBGOPTS = -g
THUMBOPTS = -mcpu=$(CPU) -mthumb -mthumb-interwork -mno-tpcs-frame
#-mtpcs-frame
#-mno-tpcs-frame
OPTIONS = $(THUMBOPTS) $(DBGOPTS)
WSDL_OPTS = -c -t $(SOAPTYPEMAP) -o $(SOAPSERVICE_H)

# When compiling C code, -c is mandatory.
SOAPC_OPTS = -c -I$(SOAPPATH)/import

CFLAGS = -Wall -O -mno-sched-prolog $(addprefix -D,$(CDEFS))
LDFLAGS = -nostdlib -Ttext $(ROM) -Tdata $(RAM) $(addprefix -L,$(LIBPATH))

CRT0 = $(SDKPATH)/lib/crt0.o

SOAPC = $(SOAPPATH)/soapcpp2
WSDL2H = $(SOAPPATH)/wsdl2h

CC = $(PREFIX)-gcc
LD = $(PREFIX)-gcc
AS = $(PREFIX)-gcc
AR = $(PREFIX)-ar
OBJCOPY = $(PREFIX)-objcopy
OBJDUMP = $(PREFIX)-objdump
STRIP = $(PREFIX)-strip

all:  Makefile server
#client

$(SOAPSERVICE_H): Makefile $(SOAPSERVICE)
	$(WSDL2H) $(WSDL_OPTS) $(SOAPSERVICE)

runtime:  Makefile $(SOAPSERVICE_H)
	$(SOAPC) $(SOAPC_OPTS) $(SOAPSERVICE_H) && touch runtime
	cp my$(SERVER_CFILE1) $(SERVER_CFILE1)

runtime-clean:  Makefile
	@rm -fv $(SOAP_CLEANFILES) runtime

clean:  Makefile runtime-clean
	@rm -fv $(SERVER_OFILES) $(CLIENT_OFILES)

server:  Makefile runtime $(SERVER_OFILES)

#client:  Makefile $(SOAPSERVICE_H) $(CLIENT_OFILE)

$(SERVER_OFILE1):  $(SERVER_CFILE1) Makefile
	$(CC) $(OPTIONS) $(CFLAGS) $(addprefix -I,$(INCPATH)) -o $@ -c $<

$(SERVER_OFILE2):  $(SERVER_CFILE2) Makefile
	$(CC) $(OPTIONS) $(CFLAGS) $(addprefix -I,$(INCPATH)) -o $@ -c $<
```

4.3 Makefile - libgsoap

Procedimento 17 Criação da libgsoap

```
# $Id:  Makefile,v 2.0.2.1 2006/03/30 18:32:54 devel1 Exp $
# File:  Makefile
# Module:
# Project:
# Author:  Robinson Mittmann (bob@boreste.com, bob@methafora.com.br)
# Target:
# Comment:
# Copyright(c) 2003-2005 CNX Technologies.  All Rights Reserved.

PROJECT = libgsoap

LIB = $(PROJECT).a
LIST = $(PROJECT).lst

INCPATH = ./include ../../include
LIBPATH = ../../lib
HFILES = $(wildcard ./*.h) $(wildcard ./include/*.h)
CFILES = $(wildcard ./*.c)
SFILES = $(wildcard ./*.S)
OFILES = $(CFILES:.c=.o) $(SFILES:.S=.o)
DEPFILES = $(wildcard ../../include/sys/*.h)

CDEFS = _SYSCALLS_ WITH_LEAN WITH_LEANER \
          WITH_NOIDREF HAVE_CONFIG_H WITH_SOAPDEFS_H
#         WITH_NOHTTP WITH_NOIDREF HAVE_CONFIG_H WITH_SOAPDEFS_H

CPU = arm7tdmi
DBGOPTS = -g
ARMOPTS = -mcpu=$(CPU) -mthumb-interwork -mno-apcs-frame
THUMBOPTS = -mcpu=$(CPU) -mthumb -mthumb-interwork -mno-tpcs-frame
OPTIONS = $(THUMBOPTS) $(DBGOPTS) $(addprefix -D,$(CDEFS))

SFLAGS = -Wall -D_GNU_ASSEMBLER_
CFLAGS = -Wall -O2 -mno-sched-prolog -fomit-frame-pointer
ARFLAGS = -rs
```

Procedimento 18 Criação da libgsoap - continuação

```
CC = arm-elf-gcc
LD = arm-elf-gcc
AS = arm-elf-gcc
AR = arm-elf-ar
OBJCOPY = arm-elf-objcopy
OBJDUMP = arm-elf-objdump
STRIP = arm-elf-strip

all:  Makefile lib
       cp $(LIB) $(LIBPATH)
lib:  Makefile $(LIB) $(LIST)
$(LIB): $(OFILES)
       $(AR) $(ARFLAGS) $@ $?
$(LIST): $(LIB)
        $(OBJDUMP) -w -t -d -S $(LIB) > $@
clean:
       @rm -fv $(OFILES) $(LIB) $(LIST)
%.o :  %.c Makefile $(HFILES)
        $(CC) $(OPTIONS) $(CFLAGS) $(addprefix -I,$(INCPATH)) -o $@ -c $<
%.o :  %.S Makefile
        $(AS) $(OPTIONS) $(SFLAGS) $(addprefix -I,$(INCPATH)) -o $@ -c $<
```