

UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

**FERNANDO YUJI MIYAKAWA**

**Desenvolvimento de uma Aplicação para Provedimento de  
Web Services em Redes Ad Hoc**

Trabalho de conclusão de curso apresentado como parte dos requisitos  
para obtenção do grau de Bacharel em Ciências da Computação

Orientador: **Mário Antônio Ribeiro Dantas**

**Florianópolis, maio de 2006**

**FERNANDO YUJI MIYAKAWA**

**Desenvolvimento de uma Aplicação para Provimento de  
Web Services em Redes Ad Hoc**

Trabalho de conclusão de curso apresentado como parte dos requisitos para  
obtenção do grau de Bacharel em Ciências da Computação

Banca Examinadora

---

Mário Antônio Ribeiro Dantas

Orientador

---

Rosvelter Coelho da Costa

Membro

---

Vitório Bruno Mazzola

Membro

Florianópolis, maio de 2006

## Sumário

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introdução.....</b>                            | <b>7</b>  |
| 1.1      | Objetivos Gerais.....                             | 8         |
| 1.2      | Objetivos Específicos.....                        | 8         |
| 1.3      | Justificativa ou Problema.....                    | 8         |
| <b>2</b> | <b>Computação Móvel.....</b>                      | <b>10</b> |
| 2.1      | Introdução.....                                   | 10        |
| 2.2      | Tecnologias.....                                  | 12        |
| 2.2.1    | Padrão IEEE 802.11.....                           | 12        |
| 2.2.2    | Bluetooth.....                                    | 14        |
| 2.2.3    | WAP.....  | 17        |
| 2.3      | Modelos de Comunicação.....                       | 21        |
| 2.3.1    | Modelo Cliente / Servidor.....                    | 21        |
| 2.3.2    | Modelo Peer-to-Peer.....                          | 21        |
| 2.3.3    | Modelo Agentes Móveis.....                        | 22        |
| 2.4      | Desafios.....                                     | 23        |
| 2.4.1    | Acessar Serviços em Ambientes Móveis.....         | 23        |
| 2.4.2    | Protocolos para Suporte a Computação Móvel.....   | 25        |
| 2.4.3    | Segurança.....                                    | 27        |
| <b>3</b> | <b>Redes Ad Hoc.....</b>                          | <b>29</b> |
| 3.1      | Vantagens.....                                    | 30        |
| 3.2      | Desvantagens.....                                 | 31        |
| <b>4</b> | <b>Web Services.....</b>                          | <b>32</b> |
| 4.1      | XML.....  | 33        |
| 4.2      | WSDL.....   | 38        |
| 4.3      | SOAP.....   | 41        |
| 4.4      | UDDI.....   | 42        |
| <b>5</b> | <b>Proposta do Projeto e Resultados.....</b>      | <b>45</b> |
| 5.1      | O Modelo Web Services em Redes Móveis Ad Hoc..... | 45        |
| 5.1.1    | Conexão com a Rede Fixa.....                      | 46        |
| 5.1.2    | DMF.....  | 48        |
| 5.1.3    | Funcionamento.....                                | 49        |
| 5.2      | Funcionalidades da aplicação.....                 | 52        |
| 5.2.1    | Oferta de serviço.....                            | 52        |
| 5.2.2    | Download de serviço.....                          | 52        |
| 5.2.3    | Envio do serviço desejado.....                    | 53        |
| 5.2.4    | Informações dos serviços.....                     | 53        |
| 5.2.5    | ServerRunner.....                                 | 53        |
| 5.3      | Aplicabilidade.....                               | 56        |
| 5.4      | Estudo de caso.....                               | 56        |
| 5.5      | Ambiente Experimental.....                        | 57        |
| 5.5.1    | Dispositivos Móveis.....                          | 57        |
| 5.5.2    | Rede Fixa.....                                    | 58        |
| 5.5.3    | Ferramentas, APIs e outros.....                   | 59        |
| 5.5.4    | Comunicação.....                                  | 59        |
| 5.6      | Testes e Resultados.....                          | 61        |
| <b>6</b> | <b>Conclusões e Trabalhos Futuros.....</b>        | <b>68</b> |

## LISTA DE FIGURAS

|  |    |
|--|----|
| FIGURA 2.1 Modelo de Programação WAP .....                                 | 18 |
| FIGURA 2.2 Exemplo de Rede WAP .....                                       | 19 |
| FIGURA 2.3 Pilha de Protocolos WAP.....                                    | 20 |
| FIGURA 2.4 Arquitetura baseada em multi-canais .....                       | 24 |
| FIGURA 3.1 Rede Ad Hoc .....   | 30 |
| FIGURA 4.1 Exemplo XML visualizado em browser .....                        | 37 |
| FIGURA 5.1 Arquitetura geral do modelo.....                                | 46 |
| FIGURA 5.2 Modelo utilizado .....  | 50 |
| FIGURA 5.3 Modelo Proposto detalhado .....                                 | 51 |
| FIGURA 5.4 Visão geral das funcionalidades da aplicação .....              | 54 |
| FIGURA 5.5 Tela principal da aplicação.....                                | 55 |
| FIGURA 5.6 Dispositivo móvel adotado no ambiente experimental .....        | 58 |
| FIGURA 5.7 Criação do root.....  | 62 |
| FIGURA 5.8 Oferta de serviço.....  | 63 |
| FIGURA 5.9 Chegada do wsdl e pedido de download .....                      | 63 |
| FIGURA 5.10 Chegada do serviço requisitado.....                            | 64 |
| FIGURA 5.11 Conteúdo do diretório serviço de cada dispositivo da rede..... | 64 |
| FIGURA 5.12 Resultado da pesquisa .....                                    | 66 |
| FIGURA 5.13 Endereço do serviço pesquisado .....                           | 66 |

## LISTA DE QUADROS

|  |           |
|--|-----------|
| <b>QUADRO 2.1 Padrão IEEE 802.11 .....</b>                 | <b>13</b> |
| <b>QUADRO 2.2 Comparação IEEE 802.11 x Bluetooth .....</b> | <b>16</b> |
| <b>QUADRO 4.1 Arquivo WSDL .....</b>                       | <b>40</b> |
| <b>QUADRO 5.1 Etapas do funcionamento do modelo .....</b>  | <b>50</b> |

# Resumo

Numa rede ad hoc, os nós possuem a capacidade de se comunicarem independentemente de qualquer infra-estrutura física de comunicação ou administração centralizada, na qual alguns dos dispositivos da rede fazem parte da rede de fato apenas durante a duração da sessão de comunicação, ou enquanto estiverem a uma certa proximidade do restante da rede. Assim sendo, essas redes são rapidamente estabelecidas e altamente flexíveis, podendo chegar a lugares difíceis ou até mesmo inacessíveis às redes infra-estruturadas.

A arquitetura *Web Services* surgiu com o intuito de tornar possível a interação de aplicações sobre a rede de forma padronizada, utilizando para isso padrões abertos como SOAP (*Simple Object Access Protocol*), UDDI (*Universal Description Discovery & Integration*) e WSDL (*Web Service Definition Language*), todos estes com base na linguagem XML (*eXtensible Markup Language*) a qual permite a troca de dados em ambientes heterogêneos, fornecendo interoperabilidade e extensibilidade.

Este trabalho apresenta o desenvolvimento de uma aplicação para fornecer e executar web services em redes Ad Hoc. A idéia principal é permitir que usuários dessa aplicação possam ter acesso a serviços que geralmente estão disponíveis apenas na rede fixa, ou ainda, necessitam dela no momento da sua utilização.

# Abstract

In an ad hoc network, the nodes have the capacity of communicating among themselves independently of any physical infrastructure or centralized administration, which some network devices are part of the network only during the duration of the communication session, or while they are in a proximity of the rest of the network. Such being, these networks are fastly established and very flexible, being able to arrive in difficult or even inaccessible places of infrastructured network.

The *Web Services* architecture appeared with the intention of making possible the interaction of applications over the network in a standardized form, using open standards like SOAP (*Simple Object Access Protocol*), UDDI (*Universal Description Discovery & Integration*) and WSDL (*Web Service Definition Language*), all of those based in a XML language (*eXtensible Markup Language*) which allows data exchange in heterogeneous environments, giving interoperability and extensionality.

This work shows us the development of an application to supply and to execute *web services* in Ad Hoc networks. The main idea is to allow that the user of this application could have access to services that usually are available only in a fixed network, or even, when they need it in the moment of its uses.

# 1 Introdução

Observando o grande crescimento nas áreas de comunicação celular, redes locais sem fio e serviços via satélite juntamente com o comércio de dispositivos que utilizam tais serviços, estima-se que em poucos anos, dezenas de milhões de pessoas terão um *laptop*, *palmtop* ou algum tipo de PDA (*Personal Digital Assistants*). Este crescimento permitirá, em um futuro bem próximo que informações e recursos possam ser acessados a qualquer instante e em qualquer lugar. Independente do tipo de dispositivo portátil, a maior parte desses equipamentos deverá ter capacidade de se comunicar com a parte fixa da rede e, possivelmente, com outros computadores móveis. A esse ambiente de computação dá-se o nome de computação móvel.

Este tipo de ambiente, onde os usuários móveis podem realizar comunicações sem-fio para acessar recursos distribuídos, faz parte da linha de pesquisa de Redes Móveis sem-fio. Basicamente, existem dois tipos de Redes Móveis sem-fio: as redes ad hoc e as redes infra-estruturadas.

*Web services* utilizados com a computação móvel resultará em ganhos significativos para consumidores e fornecedores de aplicações, tornando factível aos usuários dessas redes o acesso a uma gama de aplicações que geralmente estão disponíveis somente aos usuários das redes infra-estruturadas, ou ainda, exigem de tais usuários o acesso a uma rede fixa no momento da utilização do serviço requerido, mantendo os serviços disponíveis apenas dentro de uma determinada área de cobertura alcançada pelo *access point* (FARIA et al., 2004).

## 1.1 Objetivos Gerais

Têm-se como objetivos gerais o desenvolvimento de uma aplicação que possa disponibilizar a arquitetura *web services* em redes *Ad Hoc* e de maneira *Ad Hoc de facto*, ou seja, o usuário acessa o serviço sem o auxílio da rede fixa, explorando assim, em sua totalidade, as características da rede *Ad hoc*.

## 1.2 Objetivos Específicos

- Estudar o modelo proposto por (FARIA et al., 2004) para fornecer *web services* em Redes *Ad Hoc*.
- Desenvolver uma aplicação que forneça serviços em redes *Ad Hoc* de acordo com o modelo (FARIA et al., 2004).
- Analisar o funcionamento da aplicação de acordo com o fornecimento de serviços segundo modelo (FARIA et al., 2004).

## 1.3 Justificativa ou Problema

A arquitetura *web services* traz consigo a possibilidade de integração de aplicações de modo que elas troquem informações de maneira padronizada.

As redes *Ad hoc* são formadas dinamicamente, sem a necessidade de infra-estrutura fixa, chegando a lugares de difícil ou impossível acesso às redes infra-estruturadas.

Há um grande interesse da comunidade científica em disponibilizar serviços aos usuários das redes móveis. Esses usuários possuem uma grande



carência de serviços, uma vez que tais serviços encontram-se geralmente na rede fixa, ou dependem dela no momento de sua execução.

O principal diferencial da nossa pesquisa está no fato de tornar os serviços disponíveis de modo *Ad hoc*, independente da rede fixa (FARIA et al., 2004).

## 2 Computação Móvel

### 2.1 Introdução

A computação móvel vem surgindo como uma nova proposta de paradigma computacional advinda da tecnologia de rede sem fio e dos sistemas distribuídos. Nela o usuário, portando dispositivos móveis, como *palmtops* e *notebooks*, tem acesso à uma infra-estrutura compartilhada independente da sua localização física. Isto fornece uma comunicação flexível entre as pessoas e um acesso contínuo aos serviços de rede. É esperado que a computação móvel revolucione o modo como os computadores são usados hoje.

No decorrer do tempo, a evolução dos sistemas de computação (processamento em lotes, compartilhamento de tempo, redes com sistemas compartilhados, sistemas distribuídos) diminuiu o forte acoplamento do usuário aos recursos e ao ambiente computacional. Dentro desta progressão, os sistemas de computação móvel representam o próximo passo lógico na separação do usuário e do ambiente computacional. O usuário pode acessar os recursos do sistema (hardware e software) a qualquer tempo, desde que localizado dentro dos limites de uma infra-estrutura de comunicação.

O termo computação móvel não é um conceito ainda bem definido, e envolve elementos como hardware, dados, aplicações e usuários que têm a capacidade de se moverem para diferentes localizações durante o curso da computação. Portanto, dependendo do(s) elemento(s) que pode(m) se mover, têm-se diferentes cenários na computação móvel:

- o hardware pode se mover (*nomadic computing*);
- o usuário pode se mover entre um conjunto fixo de estações conectadas à rede (*wireless computing*);
- a aplicação pode se mover (*mobile computation: mobile code / mobile agent*);
- o usuário, portando um equipamento portátil (hardware), executando aplicações com dados e código móvel, se locomove (*pervasive computing*).

A evolução conjunta da comunicação sem fio e da tecnologia da Informática busca atender muitas das necessidades do mercado. Serviços celulares, redes locais sem fio, transmissão de dados via satélite, etc. A comunicação sem fio é um suporte para computação móvel, que pode ser vista como uma área da comunicação sem fio, que por sua vez explora diferentes tecnologias de comunicação que serão inseridas em ambientes computacionais fixos e móveis.

A mobilidade introduz problemas e desafios que não víamos, ou ignorávamos em ambientes fixos. Vários problemas já relativamente bem resolvidos em computação fixa, ou convencional, permanecem praticamente em aberto nos ambientes móveis. Os problemas que a mobilidade impõe as redes de computador vão desde a velocidade do canal, passando por interferências do ambiente e localização da estação móvel, até duração da bateria desta estação. Os projetos de instalação e expansão dos sistemas de comunicação móvel requerem em geral, grandes investimentos, o que torna os problemas grandes desafios técnico

e econômico a serem resolvidos. Além disso, existem novos problemas relacionados com os projetos de hardware e software devido à mobilidade dos elementos computacionais usados na computação móvel.

## **2.2 Tecnologias**

### **2.2.1 Padrão IEEE 802.11**

O padrão de comunicação IEEE 802.11 foi criado em 1999 para suportar a comunicação em redes locais sem fio, (WLANs – *Wireless Local Networks*). A especificação define uma camada de acesso ao meio, camada MAC, e diferentes camadas físicas, tornando possível acessar o meio de três formas possíveis: FHSS (*Frequency Hopping Spread Spectrum*), DSSS (*Direct Sequence Spread Spectrum*) e infravermelho. Muitas vezes o padrão 802.11 é chamado de “Ethernet sem fio”, por ser uma extensão natural do padrão *Ethernet* (IEEE 802.3) (MATEUS & LOUREIRO, 2004).

A unidade de arquitetura é um BSS (*Basic Service Set*). Um BSS é definido como um grupo de estações comunicantes sob controle de uma função de coordenação (DCF – *Distributed Coordination Function*), que é responsável por determinar quando um dispositivo pode enviar e receber dados (MATEUS & LOUREIRO, 2004).

As taxas de comunicação variam de acordo com a versão do padrão. No padrão IEEE 802.11, existem duas taxas de comunicação: 1 e 2 Mbps. Os padrões 802.11a e 802.11b alteraram a especificação para prover taxas de 5,5 e 11 Mbps (802.11b), chegando até 54 Mbps (802.11a). O 802.11a utiliza um

esquema especial de multiplexação para atingir altas taxas de comunicação, o que torna impossível a comunicação entre dispositivos 802.11a e 802.11b.

Atualmente, é comum ter uma infra-estrutura baseada no padrão 802.11 disponível para clientes em livrarias, cafeterias, e outros estabelecimentos comerciais.

|  | <b>802.11b/g</b>       | <b>802.11a/h</b>                                       |
|--|------------------------|--|
| <b>Frequência</b>                      | 2.4 GHz                | 5 GHz  |
| <b>Taxa de transmissão</b>             | 11/54 Mbps             | 54 Mbps  |
| <b>Raio de transmissão</b>             | 100 m                  | 50 m   |
| <b>MAC</b>                             | CSMA/CA                | CSMA/CA  |
| <b>Modulação</b>                       | DSSS/OFDM              | OFDM   |
| <b>Autenticação</b>                    | Sim                    | Sim  |
| <b>Criptografia</b>                    | 128-bit RC4            | 128-bit RC4  |
| <b>Roaming</b>                         | Sim                    | Sim  |
| <b>Suporte a conexões a rede fixa</b>  | Ethernet               | Ethernet   |
| <b>Gerenciamento</b>                   | 802.11 MIB             | 802.11 MIB   |
| <b>Controle da taxa de transmissão</b> | Adaptativo             | Adaptativo   |
| <b>Uso</b>                             | Extensão de LAN e SOHO | Padrão americano para redes de alta velocidade sem fio |

**QUADRO 2.1 Padrão IEEE 802.11**

### 2.2.2 Bluetooth

A tecnologia *wireless Bluetooth* é um padrão de fato e uma especificação para enlaces entre dispositivos móveis, tais como computadores pessoais, telefones celulares, usando ondas de rádio de curto alcance. É dirigida pelo grupo industrial *Bluetooth Special Interest Group* (SIG) formado pelas empresas: Intel, IBM, Ericsson, Toshiba, Nokia, Lucent, entre outras.

A proposta da tecnologia *Bluetooth* é habilitar os usuários para a conexão de uma grande variedade de dispositivos de computação e telecomunicações de maneira simples e fácil, sem a necessidade de carregar, comprar ou conectar cabos. Está globalmente disponível e tem compatibilidade mundial, portanto, sendo um padrão global para conectividade *wireless*. Outro fato interessante é que a tecnologia é uma especificação aberta, o que significa dizer que qualquer empresa pode obter a especificação para desenvolvimento de seus produtos (DANTAS, 2002).

Os fabricantes garantem que as interferências são pequenas nos dispositivos *Bluetooth*. A argumentação é que a tecnologia é muito resistente às intempéries e dificuldades eletromagnéticas do mundo moderno.

As configurações das redes com a tecnologia *Bluetooth* oferecem soluções interessantes, permitindo que até sete dispositivos escravos se conectem a um dispositivo mestre. As microrredes (*piconets*) são interoperáveis e podem formar uma rede maior e flexível, na qual vários dispositivos podem entrar e sair, sem maiores prejuízos para o conjunto. A rede *Bluetooth* tem uma

largura de banda de aproximadamente 700 Kbps, podendo atingir os 2 Mbps mediante algumas adaptações.

O *Bluetooth* é uma resposta para a necessidade de conexão *wireless* de equipamentos em distâncias curtas nas seguintes áreas (DANTAS, 2002):

- Pontos de acesso de dados e voz.
- Substituição de cabos.
- Redes ad hoc.
- A transmissão ocorre via rádio, na frequência de 2,4 GHz.
- A especificação engloba tanto o hardware quanto o software.

A arquitetura do *Bluetooth* é dividida pelo SIG em:

- *Core* – é o conjunto de especificações que determina como a tecnologia funciona.
- *Profile* – define como se dará interoperabilidade aos equipamentos usando o core.

|  | <b>802.11b/g</b>       | <b>802.11a/h</b>                                       | <b>Bluetooth</b>  |
|--|------------------------|--|---|
| <b>Frequência</b>                      | 2.4 GHz                | 5 GHz  | 2.4 GHz   |
| <b>Taxa de transmissão</b>             | 11/54 Mbps             | 54 Mbps  | 1 Mbps  |
| <b>Raio de transmissão</b>             | 100 m                  | 50 m   | 10 m  |
| <b>MAC</b>                             | CSMA/CA                | CSMA/CA  | Detecção de Colisão   |
| <b>Modulação</b>                       | DSSS/OFDM              | OFDM   | FHSS  |
| <b>Autenticação</b>                    | Sim                    | Sim  | Sim   |
| <b>Criptografia</b>                    | 128-bit RC4            | 128-bit RC4  | 128-bit SAFER+  |
| <b>Roaming</b>                         | Sim                    | Sim  | Não   |
| <b>Suporte a conexões a rede fixa</b>  | Ethernet               | Ethernet   | Ethernet  |
| <b>Gerenciamento</b>                   | 802.11 MIB             | 802.11 MIB   | Nenhum  |
| <b>Controle da taxa de transmissão</b> | Adaptativo             | Adaptativo   | Não   |
| <b>Uso</b>                             | Extensão de LAN e SOHO | Padrão americano para redes de alta velocidade sem fio | Substituição de cabos de conexão entre os periféricos e as estações de trabalho |

**QUADRO 2.2 Comparação IEEE 802.11 x Bluetooth**



### 2.2.3 WAP

É uma arquitetura de protocolos que permite que dispositivos móveis executem aplicações através de redes *wireless*, fazendo acesso aos serviços disponíveis no ambiente Internet. Exemplos destes dispositivos são os telefones celulares, PDAs e *paggers*. Nasceu no final dos anos 90, como uma iniciativa das empresas Motorola, Ericsson, Nokia e Unwired Planet.

Apresenta características fundamentais como (DANTAS, 2002):

- Utilização de um modelo de programação semelhante ao da WWW convencional.
- Um ambiente WWW especifica alguns mecanismos que podem ser utilizados na construção do ambiente.
- Uso do WML (*Wireless Markup Language*), que é uma linguagem de criação de serviços similar ao HTML na WWW, mas desenvolvida para ser utilizável por dispositivos móveis com pouca memória.
- Utilização da linguagem *WMLScript*, que é uma linguagem de scripts baseada no *JavaScript* e que tem uma orientação lógica procedural.
- Uso do WTA (*Wireless Telephony Application*) e WTAI (*Wireless Telephony Application Interface*), que caracterizam um conjunto de extensões específicas para serviços de telefonia e de interface de programação.
- Arquitetura de protocolos organizada em pilha.

WAP utiliza a tecnologia *Proxy* para conexão entre o domínio *wireless* e a WWW. O *Proxy WAP* é composto das seguintes funcionalidades (DANTAS, 2002):

- **Gateway de Protocolos**, que traduz "*requests*" provenientes da pilha de protocolos WAP para a pilha de protocolos WWW (HTTP e TCP/IP);
- **Codificadores e Decodificadores de Conteúdos**. Codificadores de conteúdo codificam o conteúdo WAP (informação WML) em uma forma compacta a fim de reduzir o volume de dados a ser transmitido na rede *wireless*. Decodificadores fazem o inverso.

O *Proxy WAP* permite que conteúdo e aplicações possam se hospedar em servidores WWW convencionais, se beneficiando de toda tecnologia de internet existente (figura 2.1). O *Proxy WAP* utiliza o protocolo HTTP 1.1 para comunicação com o servidor WWW.

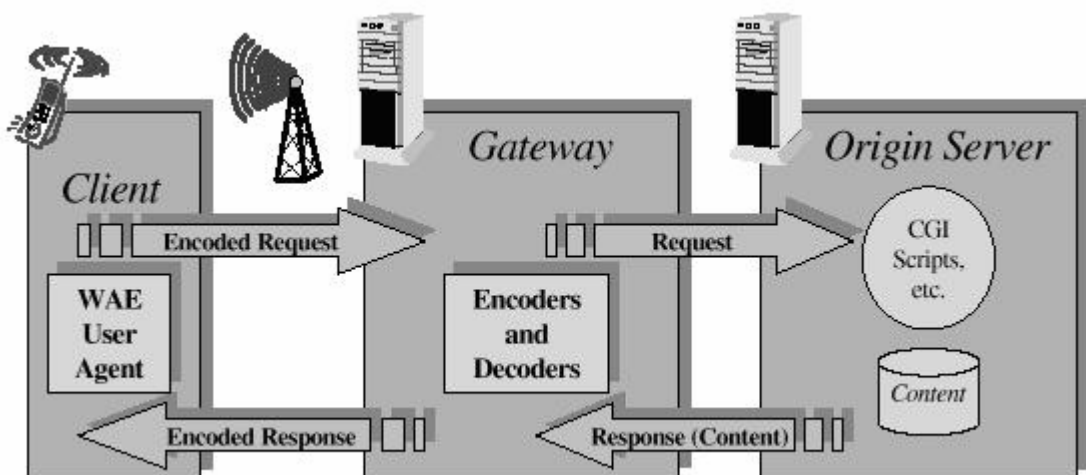


FIGURA 2.1 Modelo de Programação WAP

Neste contexto, os clientes podem utilizar URLs distintas para acessar HTML ou WML ou utilizar um único URL servindo conteúdos HTML e WML, que serão escolhidos de acordo com o *browser* requisitante.

Um exemplo de rede WAP é apresentado na figura 2.2. Nesta figura o cliente WAP (o terminal móvel) comunica com o servidor *Proxy* para acessar a internet e diretamente com o servidor WTA quando utiliza serviços de telefonia.

Ainda na figura 2.2, nota-se duas formas de acesso ao servidor WWW. Se este servidor apresentar conteúdo na forma WAP (ou seja, WML) este conteúdo é passado diretamente ao servidor *Proxy*. Caso o conteúdo seja WWW (*internet* convencional) será necessário passar por um filtro conversor HTML para WML.

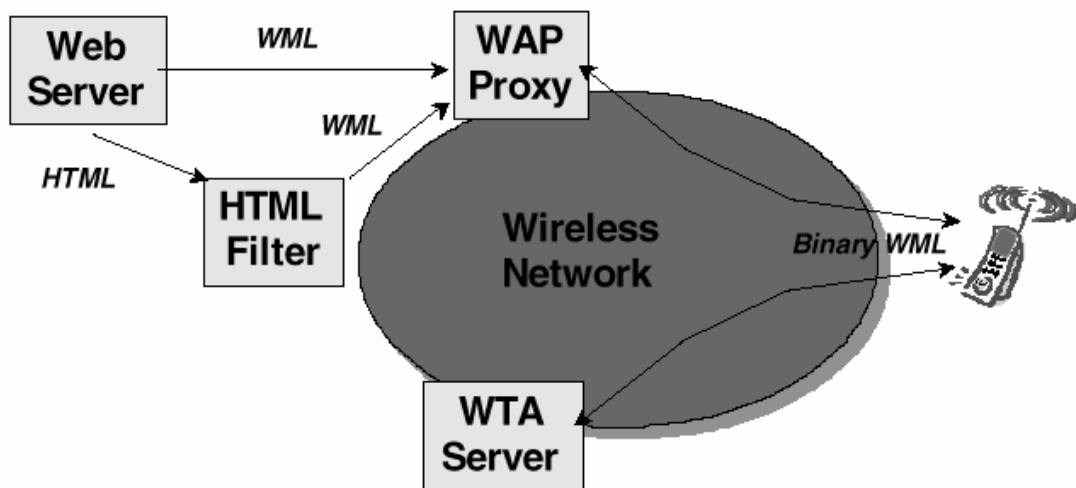
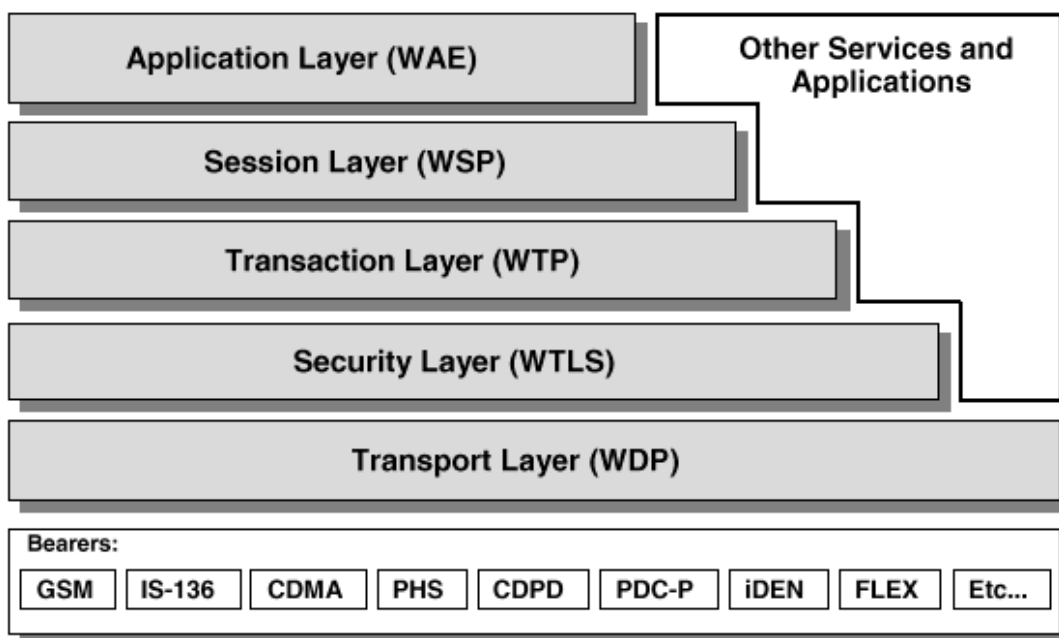


FIGURA 2.2 Exemplo de Rede WAP

A especificação da arquitetura de protocolos WAP tem como objetivo apresentar o sistema assim como a pilha de protocolos para cumprir os objetivos do WAP Forum.

A arquitetura WAP provê um ambiente escalável e extensível para desenvolvimento de aplicações para dispositivos de comunicação móveis. A figura 2.3 apresenta a pilha de protocolos WAP.



**FIGURA 2.3 Pilha de Protocolos WAP**

A arquitetura em camadas WAP possibilita que outros serviços e aplicações utilizem serviços através de um conjunto de interfaces bem definidas. Em outras palavras, aplicações externas (como correio eletrônico,

serviços de calendário ou *phonebook*) podem acessar qualquer uma das camadas diretamente.

As camadas WTP, WTLS e WDP do modelo acima apresentado possuem funções que se enquadram na camada 4 (camada de transporte) do modelo OSI (*Open System Interconnection*).

## **2.3 Modelos de Comunicação**

Existem diversos modelos de comunicação usados pelas aplicações em sistemas distribuídos. Dentre eles, destacam-se três principais modelos: o tradicional modelo cliente/servidor, modelo *peer-to-peer* e modelo agentes móveis, os quais, demonstramos a seguir (MATEUS & LOUREIRO, 1998).

### **2.3.1 Modelo Cliente / Servidor**

Em uma rede móvel *wireless*, o usuário portando um dispositivo móvel representa o cliente. Este cliente requisita informações de um ou mais servidores que normalmente estão na rede fixa. Em alguns casos, o mecanismo de replicação de servidor é utilizado para aumentar a disponibilidade do serviço.

### **2.3.2 Modelo Peer-to-Peer**

Neste modelo, os dispositivos agem tanto como clientes quanto como fornecedores para seus pares, permitindo que os dispositivos trabalhem de

modo cooperativo. Este modelo é adotado em situações em que a ligação com a rede fixa não é necessária.

### **2.3.3 Modelo Agentes Móveis**

Os agentes, com o intuito de realizar sua tarefa podem migrar de uma estação para outra, disparar a execução de novos agentes na rede ou ainda, interagir com agentes existentes. Adicionalmente, o agente executa sua tarefa de maneira autônoma e independente da aplicação que o invocou. Ao terminar a tarefa o agente retorna o resultado para a aplicação solicitante. Uma importante característica dos agentes é sua capacidade de operar em modo desconectado.

Dessa forma, depois de executar uma determinada operação, o agente aguarda até que o dispositivo esteja conectado novamente para enviar o resultado.

Os agentes são capazes de rodar em ambientes heterogêneos, o que permite a migração de uma estação para outra. Essa é uma importante característica no ambiente móvel *wireless*, já que, tais ambientes são compostos por equipamentos de diferentes fabricantes que executam sobre plataformas de software distintas. Os principais problemas do modelo agentes móveis estão relacionados às questões de segurança: autenticação, privacidade, etc. Exemplo: agentes maliciosos, ou ainda, nodos maliciosos.

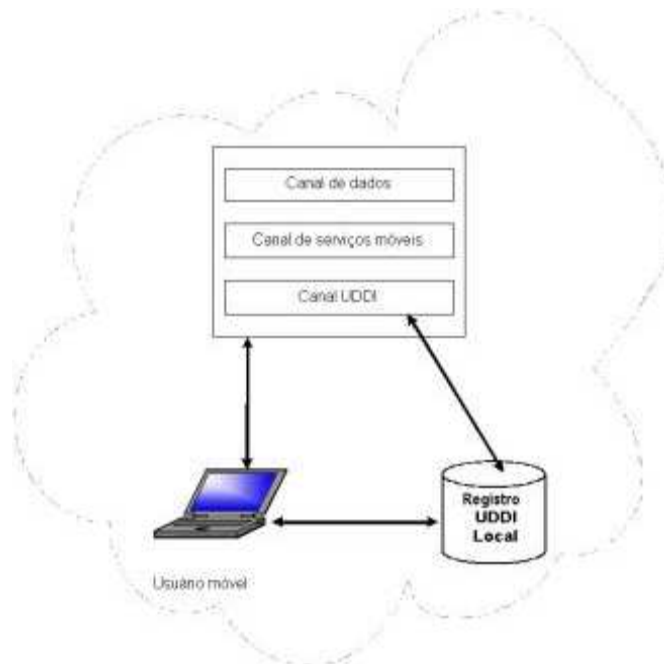
## 2.4 Desafios

### 2.4.1 Acessar Serviços em Ambientes Móveis

Visto que as peculiaridades dos ambientes móveis obrigam que sejam reformulados os mecanismos que foram projetados para os ambientes fixos para que o acesso e a consequente utilização dos serviços móveis sejam realizados de forma satisfatória, devemos pesquisar meios que torne eficiente o acesso a esses serviços, assunto esse que engloba um pouco de cada um dos itens mencionados anteriormente. Esses mecanismos deverão de algum modo lidar com os desafios que encontramos nos ambientes móveis quando tentamos portar, por exemplo, a arquitetura *web services* ao mundo *wireless* de serviços móveis.

Seguindo essa linha descrevemos a seguir dois trabalhos na área de acesso a serviços móveis.

Os autores (YANG et al., 2003) descrevem uma estrutura para organizar e acessar *web services* eficientemente em ambientes *broadcast*. Foi definido um modelo multicanal para transmitir as informações sobre *M-Services* dentro de uma determinada área geográfica. O Canal UDDI inclui informação de registro sobre os *m-services*. O canal *m-services* possui a descrição e o código executável de cada *m-service*. O canal de dados contém os dados atuais necessários para execução do *m-service* no dispositivo móvel. A figura 4.2 apresenta a interação do usuário móvel com esses canais e do registro local UDDI com o canal UDDI com o intuito de manter os serviços atualizados.



**FIGURA 2.4** Arquitetura baseada em multi-canal

A pesquisa apresenta três modos para acessar *m-services*: *at-hand*, *on-demand* e *broadcast*. No modo *broadcast*, no qual foi fundamentada a pesquisa, um *service broker* periodicamente transmite os serviços disponíveis sobre o canal *wireless*. Clientes escutam o canal, identificando o serviço de interesse e baixam o serviço para execução local. Este modo mostra-se o mais adequado em ambientes onde temos um grande número de clientes em movimentação, a performance não depende do número de clientes e também não sobrecarrega o servidor com requisições de clientes.

O segundo trabalho têm seu foco principal no armazenamento de *web services* em redes *Ad hoc* para tornar os serviços mais acessíveis a dispositivos móveis. O autor (FRIEDMAN, 2002) define que cada serviço deverá ter um único *proxy* dentro de cada rede *Ad hoc* no qual o serviço está sendo usado, ou seja, todas as invocações a um determinado serviço dentro desta rede, deverão ser feitas a este *proxy*.



O autor lida com alguns desafios como, por exemplo, escolher a melhor localização para o *proxy*, garantir que pelo menos um dispositivo atuará como *proxy*, além de garantir que ele continuará a atuar como *proxy* até que outro dispositivo seja escolhido, no caso de desistência, por exemplo.

Outra consideração importante assumida pelo autor é que o primeiro nodo a acessar um serviço, se tornará o *proxy*. Caso o nodo fique sobrecarregado ele poderá repassar alguns serviços para outro nodo.

## **2.4.2 Protocolos para Suporte a Computação Móvel**

Computadores na arquitetura TCP/IP usada na Internet possuem um endereço IP que determina o roteamento de pacotes a serem entregues a um destinatário. Por trás deste conceito está o fato que os computadores são fixos e o endereço determina a localização de um computador em relação ao restante da rede. No entanto, no caso de computadores móveis, isto não é válido já que a localização de uma unidade móvel muda. Se o endereço associado com o computador móvel permanece o mesmo, independente de sua localização, então o endereço não pode ser usado para rotear pacotes IP, já que pode não representar a localização atual de um computador móvel. Por outro lado, se um computador móvel possui um endereço, que é função de sua posição, então todas as outras entidades (computadores, processos, aplicações, etc.) em contato com esse computador precisam ser informadas de mudanças no endereço. No caso de redes com muitos computadores móveis ou composta de computadores com alta taxa de mobilidade, esta estratégia

possui sérios problemas de desempenho, visto que uma grande quantidade de informação deve ser difundida na rede para notificar todos os elementos dos novos endereços dos computadores.

Já nas estratégias com endereçamentos fixos, cada computador possui um endereço único de comunicação. Neste caso, quando um computador deseja enviar um pacote para uma unidade móvel basta utilizar o endereço conhecido. Nesta estratégia é responsabilidade da camada de rede redirecionar o pacote transmitido até o seu endereço final.

Uma das técnicas é utilizar mensagens de broadcast para localizar o computador móvel e depois entregar o pacote. Esta abordagem possui a desvantagem de sobrecarregar a rede de comunicação.

Outra abordagem é a utilização de uma central de informação, responsável por conhecer a localização física de cada computador na rede. Neste caso, basta consultar o centro de informação para saber a localização corrente do computador móvel. A principal desvantagem desta abordagem é que este centro de informação passa a ser um ponto de falha em potencial na rede, uma vez que a falha desse elemento implica na falha de todo o sistema de comunicação. Esse problema pode ser minimizado com a replicação de centros (MATEUS & LOUREIRO, 2004).

Uma alternativa para esta abordagem é o conceito de *home base* de um computador móvel, ou seja, todo computador móvel possui uma estação base responsável pelo redirecionamento de suas mensagens. Neste caso, toda vez que um computador desejar enviar um pacote para um computador móvel basta que o pacote seja enviado para a sua *home base* que se encarregará de

redirecionar o pacote para o endereço físico onde se encontra o computador móvel no momento (MATEUS & LOUREIRO, 2004).

Nesta abordagem, toda vez que o computador móvel alterar o seu ponto de conexão na rede, é necessário informar a sua estação base da sua nova localização. Esta solução está sendo utilizada pelo protocolo IP Móvel, com o objetivo de adaptar a versão existente do protocolo IP para o ambiente de computação móvel. A versão atual do IP Móvel se baseia no protocolo IPv4 (IP versão 4). No entanto, um grupo de trabalho da IETF (*Internet Engineering Task Force*) está adaptando este protocolo para poder trabalhar com a versão mais nova do protocolo IP ou IPv6, sendo que no momento já existe uma versão *draft* da nova especificação do IP Móvel.

### 2.4.3 Segurança

Não poderíamos deixar de mencionar o assunto segurança, indispensável para qualquer ambiente computacional, seja ele móvel ou fixo. Esse é também um tema bastante pesquisado nas redes móveis.

O trabalho dos autores (WANG et al., 2003), apresenta uma técnica para o controle de acesso seguro e flexível para *M-Services* baseado em RBAC (controle de acesso baseado em papéis). A arquitetura para o controle de acesso é formada por algumas entidades, um TCC (*trusted credential center*), um *trusted authentication*, um *registration center* (TARC), e um mecanismo baseado em *ticket* seguro para o acesso a 46 serviços.

Nessa arquitetura, *tickets* são emitidos pela entidade TCC e estes, carregam informações de autorização, necessárias para a utilização dos serviços. Os *tickets* podem especificar regras de controle de acesso para

diferentes tipos de serviços. Quando um usuário apresenta um *ticket* para o fornecedor do serviço, ele verifica a validade do *ticket* e o serviço é fornecido baseado nos privilégios de controle de acesso.

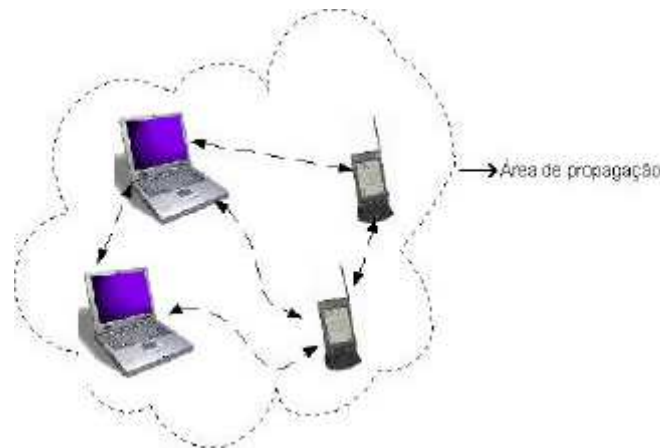
### 3 Redes Ad Hoc

O termo "*ad hoc*" é geralmente entendido como algo que é criado ou usado para um problema específico ou imediato. Do latim, *ad hoc*, significa literalmente "para isto", um outro significado seria "apenas para este propósito", e dessa forma, temporário. Contudo, "*ad hoc*" em termos de "redes *ad hoc* sem fio" significa mais que isso.

Numa rede *ad hoc*, ou MANETs (*Mobile Ad hoc Network*), os nós ou nodos possuem a capacidade de se comunicarem independentemente de qualquer infra-estrutura física de comunicação ou administração centralizada, criando uma rede "*on the fly*", na qual alguns dos dispositivos da rede fazem parte da rede de fato apenas durante a duração da sessão de comunicação, ou, no caso de dispositivos móveis ou portáteis, enquanto estiverem a uma certa proximidade do restante da rede. Assim sendo, essas redes são rapidamente estabelecidas e altamente flexíveis, podendo chegar a lugares difíceis ou até mesmo inacessíveis às redes infra-estruturadas. São indicadas principalmente em situações onde não se pode, ou não faz sentido, instalar uma rede fixa (SADOK & D'OLIVEIRA, 2001).

Dois nós que estão dentro da área de cobertura da rede, podem se comunicar diretamente, chamada de comunicação único salto (*hop*). Devido ao raio de transmissão das redes, múltiplos saltos (*hops*) podem ser necessários para efetuar a troca de dados entre dois nós que não estejam dentro da mesa área de transmissão. Assim, cada nó atua tanto como roteador quanto como um *host*, participando da descoberta e manutenção de rotas para outros nós.

Outras características incluem um modo de operação ponto a ponto distribuído, e mudanças relativamente freqüentes na concentração dos nós da rede. A responsabilidade por organizar e controlar a rede é distribuída entre os próprios terminais.



**FIGURA 3.1 Rede Ad Hoc**

Várias vantagens e desvantagens podem ser citadas ao se comparar redes *ad hoc* com redes infra-estruturadas. Alguns exemplos são (PINHEIRO, 2005):

### **3.1 Vantagens**

- Instalação rápida: redes *ad hoc* podem ser estabelecidas dinamicamente em locais onde não haja previamente uma infra-estrutura instalada;
- Tolerância à falhas: a permanente adaptação e reconfiguração das rotas em redes *ad hoc* permitem que perdas de conectividade entre os nós

possam ser facilmente resolvidas desde que uma nova rota possa ser estabelecida;

- Conectividade: dois nós móveis podem se comunicar diretamente desde de que cada nó esteja dentro da área de alcance do outro;
- Mobilidade: esta é uma vantagem primordial com relação às redes fixas.

### 3.2 Desvantagens

- Roteamento: a mobilidade dos nós e uma topologia de rede dinâmica contribuem diretamente para tornar a construção de algoritmos de roteamento um dos principais desafios em redes *ad hoc*;
- Localização: uma questão importante em redes *ad hoc* é a localização de um nó, pois além do endereço da máquina não ter relação com a posição atual do nó, também não há informações geográficas que auxiliem na determinação do posicionamento desse nó;
- Taxa de erros: a taxa de erros associada a enlaces sem-fio é mais elevada quando comparada aos enlaces em redes estruturadas;
- Banda passante: com cabeamento convencional, a banda passante pode chegar a 1Gbps. Nos enlaces via redes *wireless* temos taxas de até 2Mbps tipicamente.
- Energia escassa: a maioria dos nodos de uma MANET usa baterias como suprimento de energia. Para estes nodos o critério de projeto, mais importante, deve ser a conservação de energia. À medida que os nodos esgotam a capacidade de suas baterias eles deixam de colaborar no roteamento podendo inclusive ocasionar o particionamento da rede.

## 4 Web Services

*Web services* é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato *XML*.

*Web Services* usam tecnologias programáveis e reutilizáveis que aproveitam a flexibilidade da *Internet*. Com eles é possível ter uma infinidade de aplicativos conectados em rede, mesmo rodando em plataformas diferentes, fornecendo informações a todos os seus clientes, parceiros e funcionários. Baseiam-se num conjunto de padrões abertos, incluindo SOAP, WSDL e UDDI. O W3C e o OASIS são as instituições responsáveis pela padronização dos *Web Services*. Empresas como IBM e Microsoft, duas das maiores do setor de tecnologia, apóiam o desenvolvimento deste padrão (CHAPPELL & JEWELL, 2002).

Ainda, segundo os autores (CHAPPELL & JEWELL, 2002), *Web Services* permitem que a integração de sistemas seja realizada de maneira compreensível, reutilizável e padronizada. É uma tentativa de organizar um cenário cercado por uma grande variedade de diferentes aplicativos, fornecedores e plataformas. A seguir, serão apresentados os padrões abertos de forma mais detalhada.



## 4.1 XML

XML, *eXtensible Markup Language*, é um método padrão para se representar dados proposto pelo W3C (*World Wide Web Consortium*) a fim de atender as necessidades de comunicação entre sistemas (principalmente *Web*) fornecendo uma identificação flexível para todo o tipo de informação.

É uma linguagem de marcação de dados que provê um formato para descrever dados estruturados. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca através de múltiplas plataformas (W3C, 2004, C).

A cada dia os profissionais de TI vêem a importância da utilização do XML em suas aplicações. Muitos acreditavam que ela substituiria o HTML, *Hyper Text Markup Language*, mais isso é um erro, pois o XML foi desenvolvido para manipular dados e não mostrar os dados como no caso do HTML.

O XML permite a definição de um número infinito de *tags*. Enquanto no HTML, se as *tags* podem ser usadas para definir a formatação de caracteres e parágrafos, o XML provê um sistema para criar *tags* para dados estruturados.

Um elemento XML pode ter dados declarados como sendo preços de venda, taxas de preço, um título de livro, a quantidade de chuva, ou qualquer outro tipo de elemento de dado. Como as *tags* XML são adotadas por *intranets* de organizações, e também via *Internet*, haverá uma correspondente habilidade em manipular e procurar por dados independentemente das aplicações onde os quais são encontrados. Uma vez que o dado foi

encontrado, ele pode ser distribuído pela rede e apresentado em um *browser*, ou então esse dado pode ser transferido para outras aplicações para processamento futuro e visualização.

O XML provê um padrão que pode codificar o conteúdo, as semânticas e os esquemas para uma grande variedade de aplicações desde simples até as mais complexas, dentre elas (W3C, 2004, C):

- Um simples documento.
- Um registro estruturado tal como uma ordem de compra de produtos.
- Um objeto com métodos e dados como objetos *Java* ou controles *ActiveX*.
- Um registro de dados. Um exemplo seria o resultado de uma consulta a bancos de dados.
- Apresentação gráfica, como interface de aplicações de usuário.
- Entidades e tipos de esquema padrões.
- Todos os *links* entre informações e pessoas na *web*.

Uma característica importante é que uma vez tendo sido recebido o dado pelo cliente, tal dado pode ser manipulado, editado e visualizado sem a necessidade de reacionar o servidor. Dessa forma, os servidores têm menor sobrecarga, reduzindo a necessidade de computação e reduzindo também a requisição de banda passante para as comunicações entre cliente e servidor.

O XML ainda conta com recursos tais como folhas de estilo definidas com *Extensible Style Language (XSL)* e *Cascading Style Sheets (CSS)* para a apresentação de dados em um navegador (W3C, 1999, F). O XML separa os

dados da apresentação e processo, o que permite visualizar e processar o dado como quiser, utilizando diferentes folhas de estilo e aplicações.

Um DTD (*Document Type Declaration*) é a definição formal da estrutura de um determinado documento XML e está normalmente armazenado num ficheiro. Construir um documento XML de acordo com um determinado DTD significa que esse documento está em conformidade e a sua estrutura é válida para esse DTD. No DTD definem-se os nomes dos elementos e atributos do documento, para além das suas características (a sua ordenação, posição relativa e conteúdo). Um documento XML bem formatado deve conter a localização do DTD ao qual está conforme.

Quanto à estrutura de um documento XML, pode-se notar que é uma árvore rotulada onde um nó externo consiste de (W3C, 2004, C):

- Dados de caracteres (uma seqüência de texto);
- Instruções de processamento (anotações para os processadores), tipicamente no cabeçalho do documento;
- Um comentário (nunca com semântica acompanhando);
- Uma declaração de entidade (simples macros);
- Nós DTD;

Um nó interno é um elemento, o qual é rotulado com:

- Um nome ou
- Um conjunto de atributos, cada qual consistindo de um nome e um valor.

Os padrões que compõem o XML são definidos pelo W3C (*World Wide Web Consortium*) e são os seguintes:

- *Extensible Markup Language* (XML) - é uma recomendação, que é vista como o último estágio de aprovação do W3C. Isso significa que o padrão é estável e pode ser aplicado à *Web* e utilizado pelos desenvolvedores de aplicações.
- *XML Namespaces* - é também uma recomendação, a qual descreve a sintaxe de *namespace*, ou espaço de nomes, e que serve para criar prefixos para os nomes de *tags*, evitando confusões que possam surgir com nomes iguais para *tags* que definem dados diferentes (XML, 2004).
- *Document Object Model* (DOM) - é uma recomendação que provê formas de acesso aos dados estruturados utilizando *scripts*, permitindo aos desenvolvedores interagir e computar tais dados consistentemente. É uma API (*Applications Programming Interface*) independente de plataforma e linguagem que é utilizada para manipular as árvores do documento XML (e HTML também) (W3C, 2004, D).
- *Extensible Stylesheet Language* (XSL) - é atualmente um rascunho. O XSL apresenta duas seções: a linguagem de transformação e a formatação de objetos (W3C, 1999, F). A linguagem de transformação pode ser usada para transformar documentos XML em algo agradável para ser visto, assim como transformar para documentos HTML, e pode ser usada independentemente da segunda seção (formatação de objetos). O *Cascade Style Sheet* (CSS) pode ser usado para XML

simplesmente estruturado, mas não pode apresentar informações em uma ordem diferente de como ela foi recebida.

- *XML Linking Language (XLL)* e *XML Pointer Language (XPointer)* - são também rascunhos. O XLL é uma linguagem de construção de *links* que é similar aos *links* HTML, sendo que é mais poderosa, porque os *links* podem ser multidirecionais, e podem existir a níveis de objetos, e não somente a níveis de páginas (W3C, 2001, G).

O XML tem por objetivo trazer flexibilidade e poder às aplicações Web.

Dentre os benefícios para desenvolvedores e usuários temos:

- Buscas mais eficientes;
- Desenvolvimento de aplicações Web mais flexíveis. Isso inclui integração de dados de fontes completamente diferentes, de múltiplas aplicações; computação e manipulação local dos dados; múltiplas formas de visualização e atualização granulares do conteúdo;
- Distribuição dos dados via rede de forma mais comprimida e escalável;
- Padrões abertos.

```
<?xml version="1.0"?>
- <funcionarios>
- <funcionario ID="2107">
  <nome>Alessandra</nome>
  <salario>1000</salario>
</funcionario>
- <funcionario ID="0200">
  <nome>Cid Onir</nome>
  <salario>700</salario>
</funcionario>
- <funcionario ID="1020">
  <nome>Bianca</nome>
  <salario>980</salario>
</funcionario>
</funcionarios>
```

FIGURA 4.1 Exemplo XML visualizado em browser

## 4.2 WSDL

WSDL (*Web Services Description Language*) é um meio de descrever as funcionalidades de um *web service*, informando as funções que cada serviço pode prestar (descrição da implementação), que informações de entradas são necessárias para que o serviço possa ser executado (descrição da interface), quais os tipos de resultados devem ser esperados (também na descrição da interface), entre outras (W3C, 2001, B).

Basicamente, quando o cliente deseja enviar uma mensagem para um determinado *Web Service*, ele obtém a descrição do serviço (através da localização do respectivo documento WSDL), e em seguida constrói a mensagem, passando os tipos de dados corretos (parâmetros, etc) de acordo com a definição encontrada no documento. Em seguida, a mensagem é enviada para o endereço onde o serviço está localizado, a fim de que possa ser processada. O *Web Service*, quando recebe esta mensagem valida-a conforme as informações contidas no documento WSDL. A partir de então, o serviço remoto sabe como tratar a mensagem, sabe como processá-la (possivelmente enviando-a para outro programa) e como montar a resposta ao cliente.

Para definir um serviço, a linguagem WSDL utiliza os seguintes elementos (W3C, 2001, B):

- **Serviço (*service*)** – os pontos finais (*endpoints*) relacionados ao serviço;
- **Porta (*port*)** – define o ponto final. Especifica uma única ligação (*binding*);

- **Ligação (*binding*)** – *binding* refere-se ao processo de associar a informação de um protocolo ou formato de dados com uma entidade abstrata, (mensagem, operação, ou *portType*);
- **Tipo de porta (*port type*)** – um conjunto abstrato de operações suportadas por uma ou mais portas;
- **Tipos (*types*)** – um container para definição de tipo de dados usando algum sistema de tipo, por exemplo, XSD, também conhecido como XML Schema;
- **Mensagem (*message*)** – uma ou mais partes (*parts*) lógicas. As partes são mecanismos flexíveis para descrever o conteúdo lógico abstrato de uma mensagem;
- **Operação (*operation*)** – descrição abstrata de uma ação suportada pelo serviço.

A linguagem WSDL define quatro operações que um ponto final (*endpoint*) suporta (W3C, 2001, B):

- **Único-caminho (*One-way*)** - o ponto final recebe uma mensagem e nada será retornado;
- **Requisição-resposta (*Request-response*)** - O ponto final recebe uma requisição e retorna uma mensagem;
- **Solicitação-resposta (*Solicit-response*)** - ponto final envia e recebe uma mensagem;
- **Notificação (*Notification*)** - O ponto final envia uma mensagem.

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <wsdl:definitions targetNamespace="http://localhost:8080/axis/Servico.jws"
03   xmlns="http://schemas.xmlsoap.org/wsdl/"
04   xmlns:apachesoap="http://xml.apache.org/xml-soap"
05   xmlns:impl="http://localhost:8080/axis/Servico.jws"
06   xmlns:intf="http://localhost:8080/axis/Servico.jws"
07   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
08   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
09   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
10   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
11   <wsdl:message name="somaRequest">
12     <wsdl:part name="valor1" type="xsd:int"/>
13     <wsdl:part name="valor2" type="xsd:int"/>
14   </wsdl:message>
15   <wsdl:message name="somaResponse">
16     <wsdl:part name="somaReturn" type="xsd:int"/>
17   </wsdl:message>
18   <wsdl:portType name="Servico">
19     <wsdl:operation name="soma" parameterOrder="valor1 valor2">
20       <wsdl:input message="impl:somaRequest" name="somaRequest"/>
21       <wsdl:output message="impl:somaResponse" name="somaResponse"/>
22     </wsdl:operation>
23   </wsdl:portType>
24   <wsdl:binding name="ServicoSoapBinding" type="impl:Servico">
25     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
26     <wsdl:operation name="soma">
27       <wsdlsoap:operation soapAction=""/>
28       <wsdl:input name="somaRequest">
29         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
30           namespace="http://DefaultNamespace" use="encoded"/>
31       </wsdl:input>
32       <wsdl:output name="somaResponse">
33         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
34           namespace="http://localhost:8080/axis/Servico.jws" use="encoded"/>
35       </wsdl:output>
36     </wsdl:operation>
37   </wsdl:binding>
38   <wsdl:service name="ServicoService">
39     <wsdl:port binding="impl:ServicoSoapBinding" name="Servico">
40       <wsdlsoap:address location="http://localhost:8080/axis/Servico.jws"/>
41     </wsdl:port>
42   </wsdl:service>
43 </wsdl:definitions>

```

**QUADRO 4.1 Arquivo WSDL**



### 4.3 SOAP

SOAP (*Simple Object Access Protocol*) é um protocolo baseado em XML para troca de informação em um ambiente distribuído e descentralizado. Produz um envelope que define um *framework* para descrever qual é a mensagem e como processá-la, regras de decodificação para mostrar tipos de dados definidos pela aplicação e aceita diversos protocolos para o transporte das mensagens, alguns deles são: HTTP, FTP, SMTP, entre outros. Através do SOAP, pode ser efetuado acesso de objeto simples permitindo aplicações para invocar métodos de objeto, ou funções, residindo em servidores remotos (W3C, 2003, I).

O protocolo SOAP é aceito como um padrão *web de facto*. O ambiente heterogêneo da *internet* exige que aplicações suportem protocolos de codificação de dados e formatos de mensagem em comum.

Caracteriza-se por ser um protocolo de comunicação de peso leve capaz de efetuar a troca de informações em ambientes distribuídos, sendo independente de linguagem e plataforma. Seus maiores objetivos são simplicidade e expansibilidade.

A especificação SOAP fornece um mecanismo para o tratamento de erros. Uma falha é usada para transmitir uma informação de erro dentro de uma mensagem SOAP e será utilizada sempre que ocorrerem erros durante o processamento das mensagens SOAP.

Uma mensagem SOAP consiste basicamente dos seguintes elementos (W3C, 2003, I):

- **Envelope** - toda mensagem SOAP deve contê-lo. É o elemento raiz do documento XML. O *Envelope* pode conter declarações de *namespaces* e também atributos adicionais como o que define o estilo de codificação (*encoding style*). Um "*encoding style*" define como os dados são representados no documento XML.
- **Header** - é um cabeçalho opcional. Ele carrega informações adicionais, como por exemplo, se a mensagem deve ser processada por um determinado nó intermediário (É importante lembrar que, ao trafegar pela rede, a mensagem normalmente passa por diversos pontos intermediários, até alcançar o destino final). Quando utilizado, o *Header* deve ser o primeiro elemento do *Envelope*.
- **Body** - Este elemento é obrigatório e contém o *payload*, ou a informação a ser transportada para o seu destino final. O elemento *Body* pode conter um elemento opcional *Fault*, usado para carregar mensagens de status e erros retornadas pelos "nós" ao processarem a mensagem.

#### 4.4 UDDI

Da mesma forma que os mecanismos de busca como *Yahoo!* e *Google* o ajudam a localizar os dados certos nas páginas *Web*, o UDDI (*Universal Description, Discovery, and Integration*) define uma maneira padrão de publicar informações e anunciar serviços.

A iniciativa de UDDI foi desenhada para permitir que as instituições realizem transações entre si de maneira mais rápida, fácil e dinâmica. Como

resultado, foi desenvolvida uma especificação para um registro central de acesso e controle, que descreve como os dados devem ser armazenados em um registro, além de definir os métodos possíveis para publicação e busca desses registros. Os dados incluem os contatos da empresa, como se comunicar com eles, uma lista de serviços disponíveis e como usá-los por meio de algum tipo de programação (OASIS, 2003).

Nesse endereço, estão todas as informações necessárias para que um *Web Service* seja utilizado. Os serviços inscritos pelas empresas em um registro de UDDI são disponibilizados para o público nos moldes do conceito de "registre uma única vez e publique em toda parte".

Resumindo, um registro de UDDI é um catálogo gigante, e, em teoria, um desenvolvedor pode procurar nele um serviço, descobrir como se conectar a ele e utilizá-lo em sua aplicação.

É baseada em XML para registrar os negócios e os *web services* oferecidos, produzindo as transações necessárias, isto é habilitar softwares para automaticamente descobrir os serviços para integrá-los.

As seguintes entidades constituem um registro de serviço (OASIS, 2003):

- **businessEntity:** descreve um negócio ou organização que fornece o *web service*;
- **businessService:** descreve uma coleção de *web services* relacionados oferecidos por uma organização descrita pelo *businessEntity*;
- **bindingTemplate:** descreve informações técnicas necessárias para usar o *web service*. Além disso, fornece suporte para

determinar um ponto de entrada técnico, suporte para serviços hospedados remotamente ou, ainda, a descrição de características únicas de um determinado serviço;

- **tModel:** possui dois principais objetivos. O primeiro é descrever um *web service* e utilizar a descrição para sua pesquisa. O segundo é tornar as descrições úteis para que se possa, a partir delas, aprender a interagir com o serviço;
- **publisherAssertion:** descreve, na visão de um *businessEntity*, o relacionamento que um *businessEntity* têm com outro *businessEntity*.

## 5 Proposta do Projeto e Resultados

Observamos ao longo da pesquisa que usuários de redes *Ad hoc* utilizam os serviços sempre com o auxílio de alguma infra-estrutura fixa, invalidando de certo modo as características intrínsecas ao seu ambiente, que seria executar esses serviços realmente de modo independente.

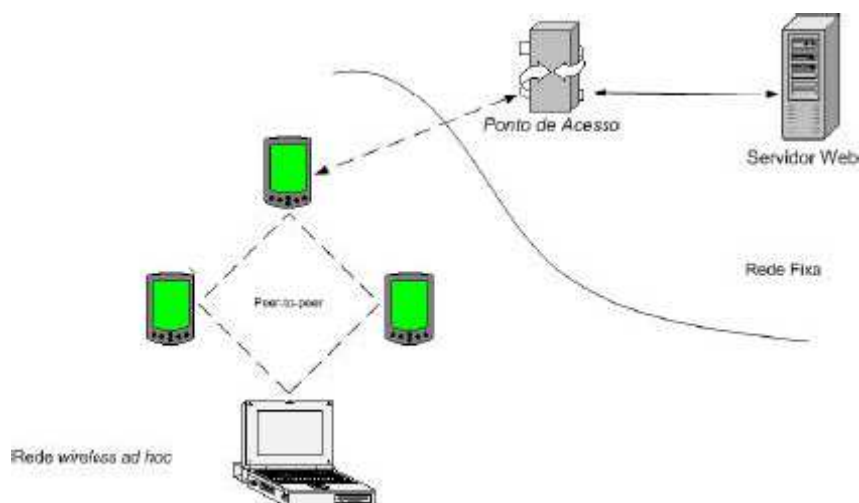
### 5.1 O Modelo Web Services em Redes Móveis Ad Hoc

Este modelo (FARIA et al., 2004), por sua vez, adota como referência a arquitetura *wireless Ad hoc network* (YUAN et al., 2002). Foi escolhido este perfil de arquitetura por ele revelar-se ideal para as redes *Ad hoc*, pois não utiliza nenhuma infra-estrutura fixa para disponibilizar o serviço aos clientes, que é um aspecto que consideramos importante no contexto deste trabalho.

Segundo os autores (FARIA et al., 2004), os dispositivos *wireless* se tornam servidores para seus pares, habilitando-os a fornecer conteúdo, roteamento do tráfego na rede, e muitos outros serviços. Por ser uma tecnologia relativamente recente, esta proposta apresenta ainda alguns aspectos a serem investigados, como as questões de desempenho e segurança.

Tendo em vista que a capacidade de armazenamento dos dispositivos móveis vem aumentando ao passar dos anos e a perda de conexão em redes *ad hoc* é freqüente, o modelo executa o *web service* localmente.

A figura 5.1 ilustra a arquitetura do modelo (Faria et al., 2004) de forma geral. Como ilustrado nessa figura, o modelo foi projetado para usuários de *notebooks* e PDA's, dando ênfase ao aspecto de autonomia e disponibilidade a aplicações onde estes requisitos sejam considerados fundamentais.



**FIGURA 5.1** Arquitetura geral do modelo

Para se ter uma melhor compreensão, será apresentado as etapas de funcionamento do modelo a seguir:

### **5.1.1 Conexão com a Rede Fixa**

O servidor conectado à rede fixa contém os módulos de software necessários para suprir um dispositivo móvel que queira tornar-se fornecedor de serviço para seus pares em uma rede *Ad hoc*. Denominamos esses módulos de módulo móvel e módulo fixo (FARIA et al., 2004).

O módulo móvel do servidor conectado à rede fixa contém um repositório UDDI privado contendo todos os serviços que esse servidor pode

disponibilizar, e mais a aplicação fornecedora e a aplicação cliente de cada serviço publicado nesse repositório UDDI.

O módulo fixo contém a máquina virtual e a base de dados. Partimos do princípio que a máquina virtual e a base de dados, ou seja, o módulo fixo estará previamente instalado nos dispositivos que formam a rede. Por tal motivo, denominamos este módulo de módulo fixo (FARIA et al., 2004).

O usuário do dispositivo móvel, ou seja, o usuário que será o fornecedor do serviço para seus pares na rede *Ad hoc* acessa a rede fixa apenas uma vez para pegar o serviço, depois disso ele poderá na rede *Ad hoc* fazer o papel do fornecedor tornando possível o acesso aos serviços sem a necessidade de qualquer outra comunicação com a rede fixa (FARIA et al., 2004).

Deve-se observar que a interação do dispositivo móvel com a rede fixa ocorre apenas neste momento, a partir disso, o usuário móvel têm autonomia suficiente para ser um fornecedor de serviços em uma rede *Ad hoc*.

Uma vez estabelecida a comunicação entre o dispositivo móvel e o servidor conectado a rede fixa, será realizada a transferência do módulo móvel. Essa comunicação poderá ser estabelecida através de duas opções. Na primeira, o dispositivo móvel é conectado fisicamente ao servidor. Já na segunda opção o dispositivo móvel acessa a rede fixa através de um ponto de acesso.

Visto que o dispositivo móvel foi suprido com o serviço de interesse, ele torna-se um DMF (Dispositivo Móvel Fornecedor), estando apto a fornecer tal serviço a seus pares em uma rede *Ad hoc*, não necessitando se conectar em qualquer outro momento ao servidor na rede fixa (FARIA et al., 2004).

### 5.1.2 DMF

Concluída a etapa anterior, o usuário do DMF dispara na rede no qual ele está localizado, uma mensagem *broadcast* oferecendo a seus pares o serviço anteriormente selecionado e adquirido através da rede fixa (FARIA et al., 2004). Limitamos os serviços à opção *read-only*, pois não lidamos com as questões de consistência dos dados, sendo isso, outra fonte de pesquisa que têm atraído a atenção de diversos pesquisadores. A consistência dos dados embora seja um tema a ser bastante explorado em serviços móveis está fora do escopo do nosso trabalho de pesquisa.

Os dispositivos que compõe a rede recebem a mensagem enviada pelo DMF e optam entre aceitar ou recusar o serviço. Os dispositivos interessados retornam a mensagem ao DMF e tornam-se nesse momento dispositivos clientes, estabelecendo assim, uma conexão com o DMF requisitando o módulo móvel (FARIA et al., 2004).

O intervalo entre essas mensagens *broadcast* enviadas à rede pelo DMF é controlado pelo próprio usuário. Dessa forma, um serviço não será oferecido em um momento inadequado, como, por exemplo, num dado instante em que ele está utilizando seu dispositivo para uma determinada tarefa que necessite total prioridade de seus recursos computacionais.

Uma vez terminada a transferência dos arquivos necessários, a execução do serviço será realizada localmente, minimizando assim, alguns dos desafios inerentes ao ambiente móvel como a freqüente perda de conexão, a energia limitada dos dispositivos, entre outros que foram mencionados anteriormente (FARIA et al., 2004). Nesse momento, o dispositivo cliente que efetuou a transferência dos módulos, além de utilizar o serviço, se desejar,



também poderá agir como DMF em qualquer rede *Ad hoc*. Dessa forma, poderemos num dado momento ter em uma mesma rede vários fornecedores e vários clientes, aumentando com isso, a disponibilidade dos serviços. No caso de um dispositivo (DMF) deixar a rede, um outro dispositivo que tenha requisitado um serviço anteriormente poderia oferecê-lo nessa mesma rede ou eventualmente, em outra rede (FARIA et al., 2004).

Devemos observar que há então dois cenários para um dispositivo móvel cliente se tornar um DMF com a única diferença que o DMF que acessar a rede fixa para a transferência do módulo móvel terá disponível o diretório UDDI com a opção de escolher entre todos os serviços que estão disponíveis enquanto que o DMF que adquiriu o módulo móvel através da rede *Ad hoc*, ou seja, através de um DMF, terá apenas os serviços adquiridos pelo DMF que se conectou com a rede fixa.

Adicionalmente, devemos complementar que o usuário terá a responsabilidade de gerenciar os recursos computacionais do seu dispositivo, controlando dessa forma, o número de vezes que ele vai oferecer o serviço à rede, ou seja, caso o usuário esteja com muitas conexões ativas ou ainda com uma carga mínima da bateria, ele não enviará mais nenhuma mensagem *broadcast* para a rede oferecendo o serviço aos seus pares até que o seu status atual volte a uma opção confiável, onde ele possa fornecer o serviço de forma satisfatória (FARIA et al., 2004).

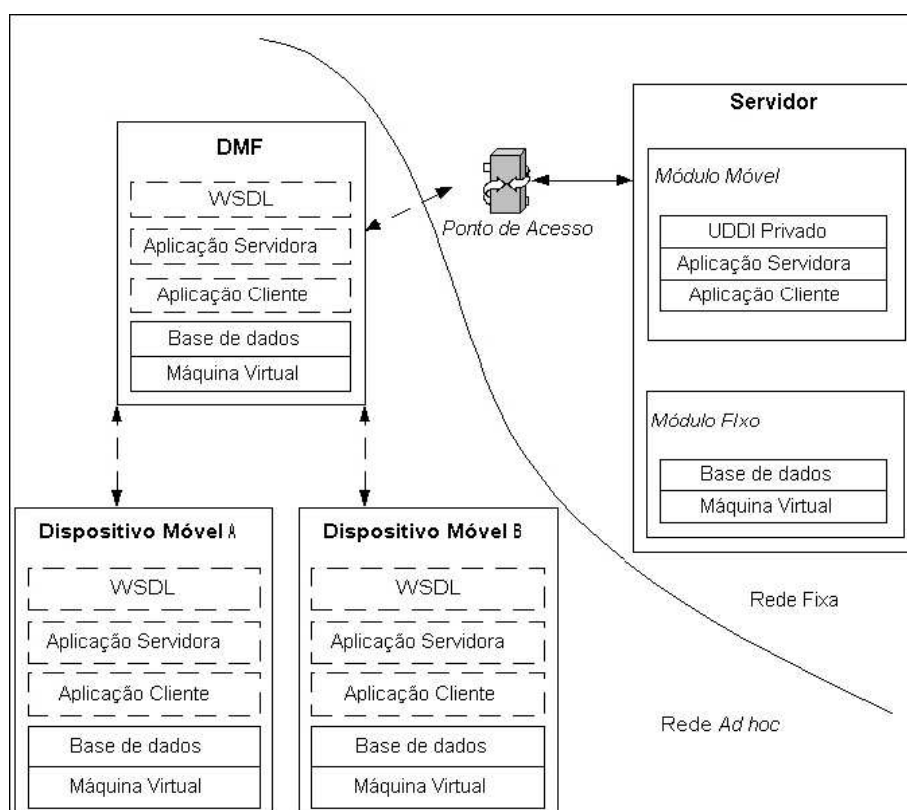
### **5.1.3 Funcionamento**

Conforme descrito no quadro 5.1, o funcionamento do modelo proposto se dá através das etapas A, B, C, D:

| Etapa | Ação   |
|-------|--|
| A     | O DMF dispara uma mensagem broadcast.  |
| B     | Os dispositivos A e B solicitam o serviço ao DMF.  |
| C     | Será estabelecida uma conexão entre DMF e dispositivos A e B para a transferência do módulo móvel. |
| D     | Execução local do serviço.   |

**QUADRO 5.1** Etapas do funcionamento do modelo

No caso de queda de conexão, ou a bateria do equipamento descarregue durante uma transferência, o cliente aguardará por uma nova mensagem *broadcast* que deverá ser enviada pelo DMF.



**FIGURA 5.2** Modelo utilizado

A figura 5.3 apresenta o modelo de forma um pouco mais detalhada. Através dessa figura observamos a interação com a rede fixa. Podemos

visualizar as duas opções existentes para a conexão com o servidor *web*, Opção A, através de um meio físico e Opção B, através do link *wireless*.

Posteriormente, o dispositivo móvel requisita o módulo móvel através de uma das opções A ou B e se torna um DMF. Concluída essa etapa, o serviço pode ser fornecido de modo *Ad hoc* aos usuários da rede.

Possuindo acesso físico ao servidor na rede fixa (opção A), ou ainda, autenticando-se na rede *wireless* (opção B), qualquer dispositivo móvel poderá se tornar um DMF.

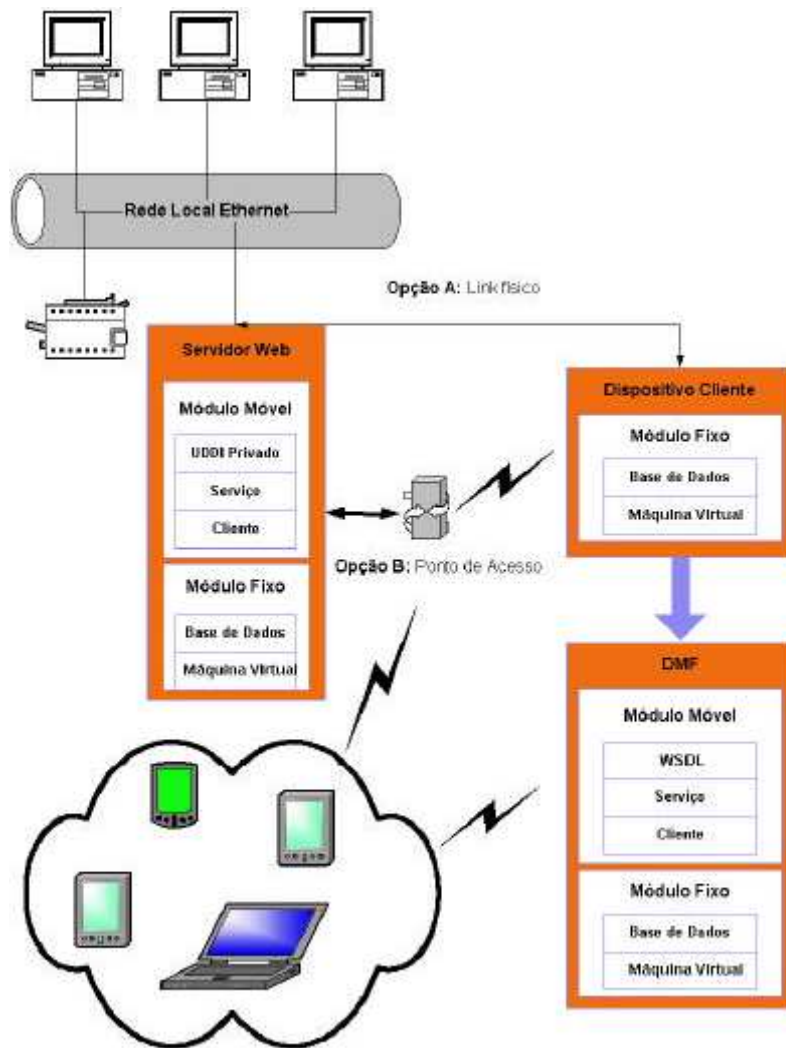


FIGURA 5.3 Modelo Proposto detalhado

Já que estamos trabalhando na camada de aplicação e não na camada de rede, o problema de roteamento não foi abordado no modelo, tendo em vista que, esse assunto não faz parte do enfoque deste trabalho.

## **5.2 Funcionalidades da aplicação**

### **5.2.1 Oferta de serviço**

Para se oferecer um serviço a outro nó, o usuário deverá escolher um serviço e confirmar, sendo assim, será enviada uma mensagem *broadcast*, que é nada mais o arquivo *wsdl* do respectivo serviço que, com isso, o usuário se informe sobre o serviço antes de baixá-lo, a todos os nós ativos da rede *Ad Hoc*. Em cada nó da rede será atualizada a lista de serviços disponíveis na rede assim que chegar a mensagem.

### **5.2.2 Download de serviço**

Através dessa funcionalidade, o usuário poderá baixar para seu dispositivo, serviços disponíveis na rede *Ad Hoc* que pode ser visualizada na lista de serviços disponíveis na rede. Ao escolher um item da lista e confirmar, será enviado o nome do serviço requisitado ao DMF (Dispositivo Móvel Fornecedor), que através deste envio, terá conhecimento do serviço propriamente dito a ser enviado de volta ao nó que pediu o serviço. É possível baixar ao mesmo tempo vários serviços, tanto de servidores diferentes como do mesmo servidor.

### 5.2.3 Envio do serviço desejado

O envio do serviço é automaticamente carregado e enviado pela aplicação, tendo em vista que na mensagem, do nó que deseja um serviço, está contido o nome do serviço desejado.

É possível também fornecer um ou mais serviços ao mesmo tempo, para diferentes clientes ou para o mesmo.

### 5.2.4 Informações dos serviços

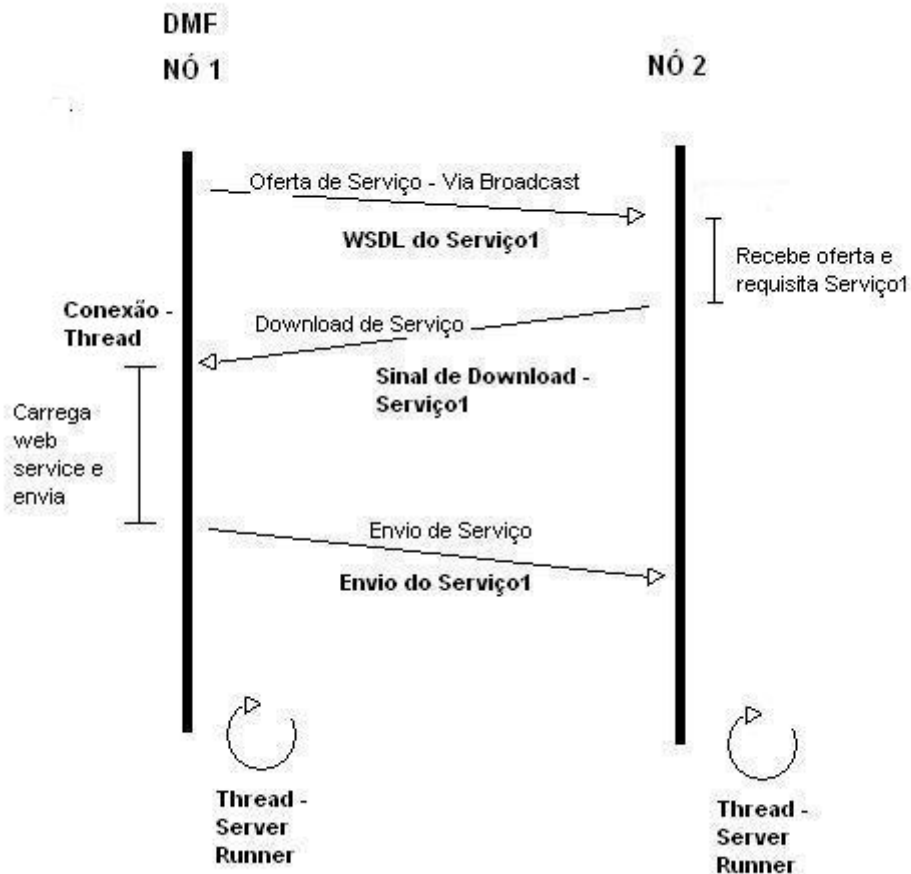
É possível obter informações de todos os serviços disponíveis na rede e localmente, já que logicamente os arquivos *wSDLs* de cada serviço encontram-se localmente e foram oferecidos em algum momento anterior. No arquivo *wSDL* pode ser encontrado informações como, o endereço do serviço no qual pode ser encontrado, protocolo de transporte utilizado, entre outras.

### 5.2.5 ServerRunner

Entidade responsável pela realização e disponibilidade das conexões.

Funciona da seguinte maneira:

- 1 quando iniciado, permanece na espera até um cliente se conectar;
- 2 cria uma thread de servidor socket e atribui a conexão que acabou de ser estabelecida à thread;
- 3 retorna ao estado de espera por uma nova conexão de um cliente.



**FIGURA 5.4** Visão geral das funcionalidades da aplicação

A partir da figura 5.4, podemos observar as funcionalidades da aplicação sendo utilizadas em dois nós dentro de uma rede *ad hoc*.

Na figura 5.5 é apresentada tela principal da aplicação, onde podem ser visualizados os serviços disponíveis na rede e os serviços que estão localmente.



**FIGURA 5.5** Tela principal da aplicação

### **5.3 Aplicabilidade**

Os tipos de serviços que se beneficiariam com a utilização do modelo são aqueles que se fazem necessários em lugares onde utilizar uma infraestrutura de rede fixa não é viável, seguindo a motivação para a utilização das redes *Ad hoc* e tornando realística a utilização de serviços sobre essas redes (Faria et al., 2004).

Alguns exemplos como, por exemplo, consulta a informações necessárias durante a pulverização de agrotóxicos em uma lavoura de arroz, ou ainda, informações sobre a área de replantio de soja. Além dessas, apresentamos um estudo de caso logo abaixo que tem como objetivo demonstrar a utilização de um serviço conforme o modelo utilizado.

### **5.4 Estudo de caso**

Uma dificuldade usual encontrada pelo corpo de bombeiros é a falta de um serviço que informe, por exemplo, onde estão localizados os hidrantes mais próximos do local onde está acontecendo um incêndio. A busca por essas informações é de extrema importância, mas toma um tempo precioso que em muitos casos, vale a vida das pessoas envolvidas numa tragédia (Faria et al., 2004).

Uma vez aberta uma chamada para atender a um incêndio, o usuário portando um dispositivo móvel faz uma conexão com servidor que contém o



serviço de localização de hidrantes e adquire o módulo móvel se tornando um DMF.

Estabelecida a rede *Ad hoc*, o serviço poderá ser oferecido por esse usuário aos seus pares na rede. A partir disso, podemos verificar que o modelo torna factível levar serviços a lugares inacessíveis a rede fixa, lugares onde a necessidade é temporária, buscando de fato, uma computação disponível a qualquer hora e em qualquer lugar.

Deve-se observar que a conexão com a rede fixa ocorre apenas em um momento, para que um dispositivo se torne um DMF, posterior e essa etapa, o serviço será oferecido e, conseqüentemente, utilizado sem o auxílio de qualquer infra-estrutura fixa.

## **5.5 Ambiente Experimental**

### **5.5.1 Dispositivos Móveis**

Como dispositivos móveis, adotou-se os equipamentos Palm Tungsten C (Figura 5.4) disponíveis no LSD (Laboratório de Sistemas Distribuídos), da UFSC, os quais são equipados com processador Intel Xscale, de freqüência 400 MHz, com 64 Mb de memória RAM, com interface WiFi (IEEE 802.11b) e infravermelho, rodando o sistema Operacional PalmOS, versão 5.2.1.

Estes equipamentos foram escolhidos pelo fato de encontrarem-se disponíveis no momento da implementação, por apresentarem uma excelente relação custo/benefício e também por representarem de modo fiel o padrão de equipamentos utilizados nas aplicações típicas de redes *Ad hoc*. A autonomia

deste equipamento em termos de energia é estimada em aproximadamente 7 horas.



**FIGURA 5.6** Dispositivo móvel adotado no ambiente experimental

### 5.5.2 Rede Fixa

Em nosso ambiente, a rede fixa utilizada é aquela instalada no LSD, a qual é composta de 6 microcomputadores com processadores AMD, rodando *Windows 2000* e *Windows XP*. A rede é baseada no padrão Ethernet (IEEE 802.3) e é conectada à rede do Departamento de Informática e de Estatística da UFSC, que possui acesso permanente à *internet*.

O servidor *web* conectado a rede fixa, contendo os módulos necessários para a disponibilizar o serviço, está equipado com processador AMD Ahtlon, com frequência de 2,4GHz, com 512 Mb de memória RAM, 40Gb de disco rígido rodando *Windows XP*.

### 5.5.3 Ferramentas, APIs e outros

No desenvolvimento do software foi utilizado J2ME Wireless ToolKit 2.2 e as APIs CLDC 1.1 (*Connected Limited Device Configuration*) e MIDP 2.0 (*Mobile Information Device Profile*) que acompanham o *Toolkit*.

Para a manipulação de dados XML foi utilizada a API KXML, que é uma API XML reduzida e quanto ao acesso a arquivos foi utilizada uma API que recentemente é suportada pelos dispositivos móveis mais modernos da atualidade, a *JSR75 FileConnection*. Ela permite às aplicações criar, ler e escrever arquivos e diretórios localizados em dispositivos móveis e em cartões de expansão.

Quanto ao protocolo de acesso à dados foi utilizado o kSOAP, versão reduzida da biblioteca SOAP convencional, que por sua vez é uma biblioteca para *web services* para ambientes *Java*, tais como J2ME. Ela apresenta características relevantes como leveza, facilidade de uso e ótima documentação.

E por fim, foi utilizado o *UDDI@SAP Test Public Registry* que é um banco de dados desenvolvido para integrar *web services* (disponível em <http://udditest.sap.com/>). Era uma das poucas instituições que ainda disponibiliza um UDDI público, já que os mais conhecidos (IBM, Microsoft, UDDI.org), no momento, não estavam mais acessíveis aos usuários.

### 5.5.4 Comunicação

A comunicação é realizada através de *sockets*. Devido a falta de padronização quanto ao mecanismo de endereçamento e tendo em vista que

esse assunto não faz parte do enfoque deste trabalho, foram atribuídos endereços IPs fixos e duas portas de comunicação:

- 5001: porta onde o *servidor socket* espera por conexões de clientes.
- 5002: porta pela qual o cliente pode se conectar com vários outros servidores.

Adicionalmente, partiu-se do princípio que os dispositivos móveis na rede *Ad hoc* não possuem disparidades quanto ao mecanismo de endereçamento utilizado. Esta ainda é uma questão de pesquisa extremamente pesquisada nas redes *Ad hoc*.

As configurações de comunicação podem ser alteradas através do arquivo *conf.txt*.

## 5.6 Testes e Resultados

Inicialmente, o roteiro dos testes era testar a aplicação em um simulador de dispositivo móvel que vem juntamente com o *J2ME Wireless Toolkit 2.2* e após a finalização da implementação e execução de testes bem sucedidos, os testes seriam passados a ser executados nos dispositivos móveis.

Nos testes da aplicação no simulador, não foi necessário incluir a biblioteca *JSR75 FileConnection* pelo fato de que a biblioteca vem juntamente com a versão 2.2 do *J2ME Wireless Toolkit*. A única biblioteca adicional incluída foi o pacote que contém a API kXML e a API kSOAP.

Uma característica da *JSR75 FileConnection* causava um certo atraso a cada teste efetuado, pois para o uso da mesma tinha que ser criado, antes da execução do *MIDlet*, um diretório *root*, onde é armazenado todos os arquivos que seriam acessados durante a execução.

Primeiramente, foi testado a oferta de serviços, para um até cinco dispositivos móveis, na qual ocorreu de forma rápida e bem sucedida, desde o envio da oferta até a chegada da oferta de serviço e atualização da lista de serviços disponíveis na rede dos dispositivos da rede.

Posteriormente, nos testes de *download* de serviço foi notado que o tempo de *download* foi adequado e, obviamente, maior que o tempo de oferta de serviço pelo fato de que o tamanho do arquivo *wsdl* ser bem menor que o tamanho dos dois arquivos (aplicação servidora e aplicação cliente) que constituem o serviço. E, além disso, observou-se a eficiência do *Server Runner*, por aumentar a disponibilidade de serviços e rapidez na abertura de conexões.

Nas figuras a seguir, pode ser acompanhado um exemplo das etapas de fornecimento de serviço em uma rede *ad hoc* com três nós:

- Na figura 5.6, é apresentado a criação do diretório *root* onde serão armazenados todos arquivos (de configuração, *wsdl*, serviços) de cada dispositivo.
- Na figura 5.7, é apresentado o momento da oferta do serviço, do dispositivo 5550000 aos outros dois na rede, 5550001 e 5550002 (envio da mensagem broadcast para os outros nós da rede);
- Na figura 5.8, é apresentado o momento da chegada do arquivo *wsdl* nos outros nós da rede e que, em seguida, farão o pedido de *download*.
- Na figura 5.9, é apresentado o momento em que os serviços requisitados chegam aos nós e, conseqüentemente, já podem atuar como DMF na rede.
- Na figura 5.10, é apresentado o conteúdo do diretório “serviços” de cada dispositivo, onde foi armazenado o serviço requisitado anteriormente pelos outros dois dispositivos (5550001 e 5550002).

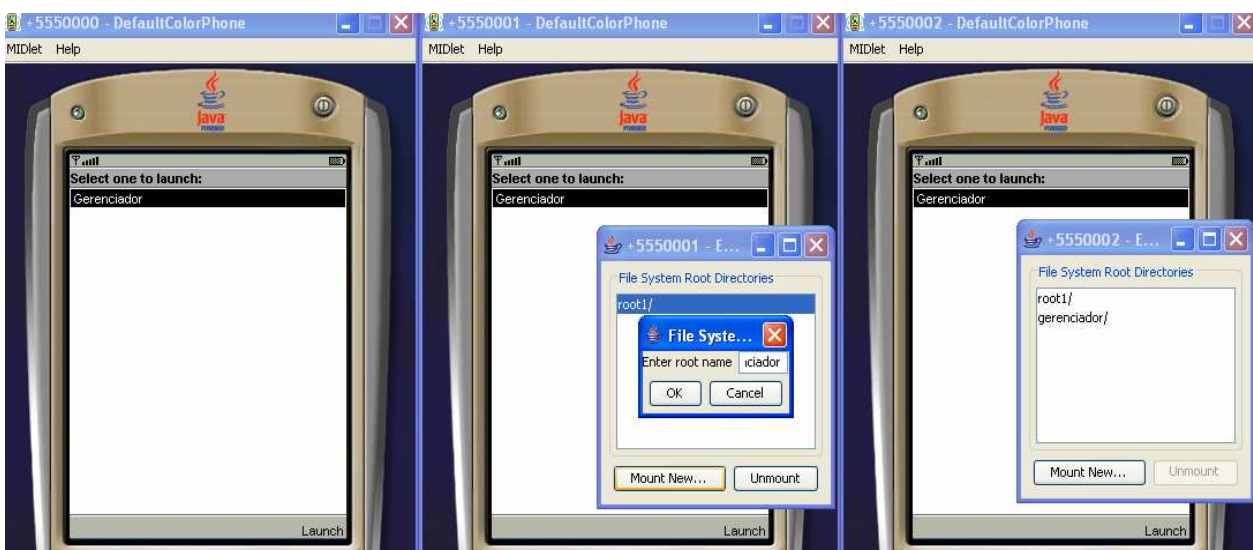


FIGURA 5.7 Criação do root



FIGURA 5.8 Oferta de serviço

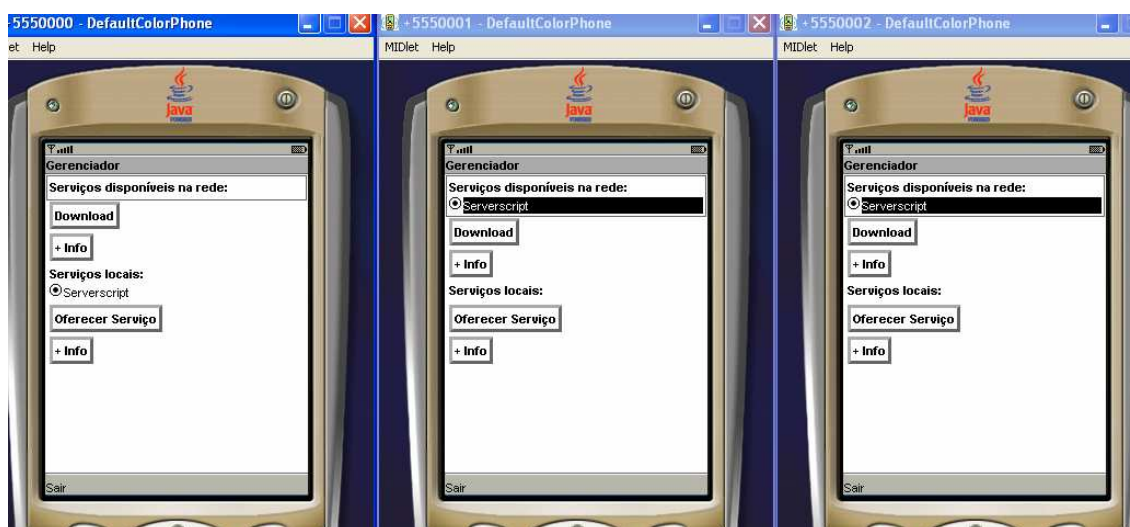


FIGURA 5.9 Chegada do wsdl e pedido de download

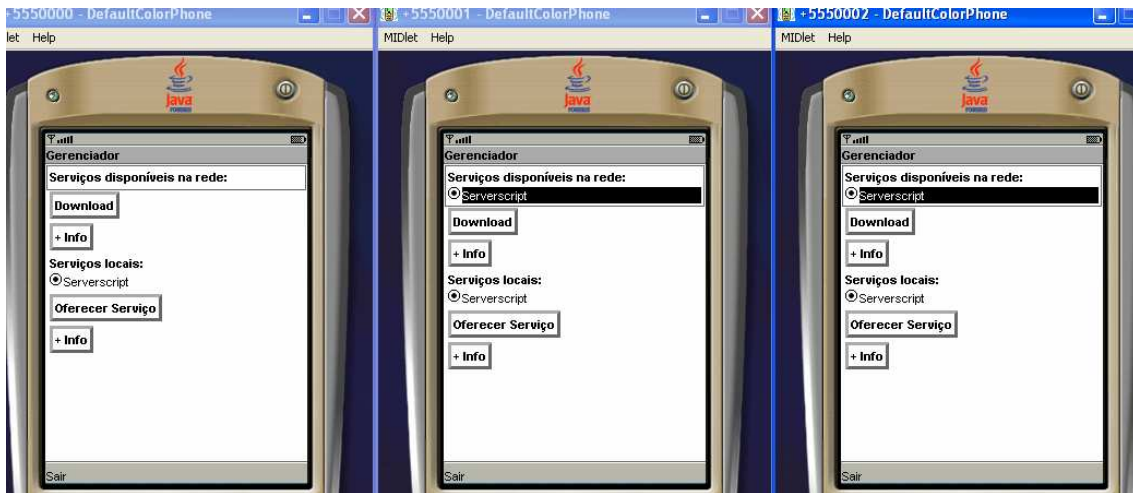


FIGURA 5.10 Chegada do serviço requisitado

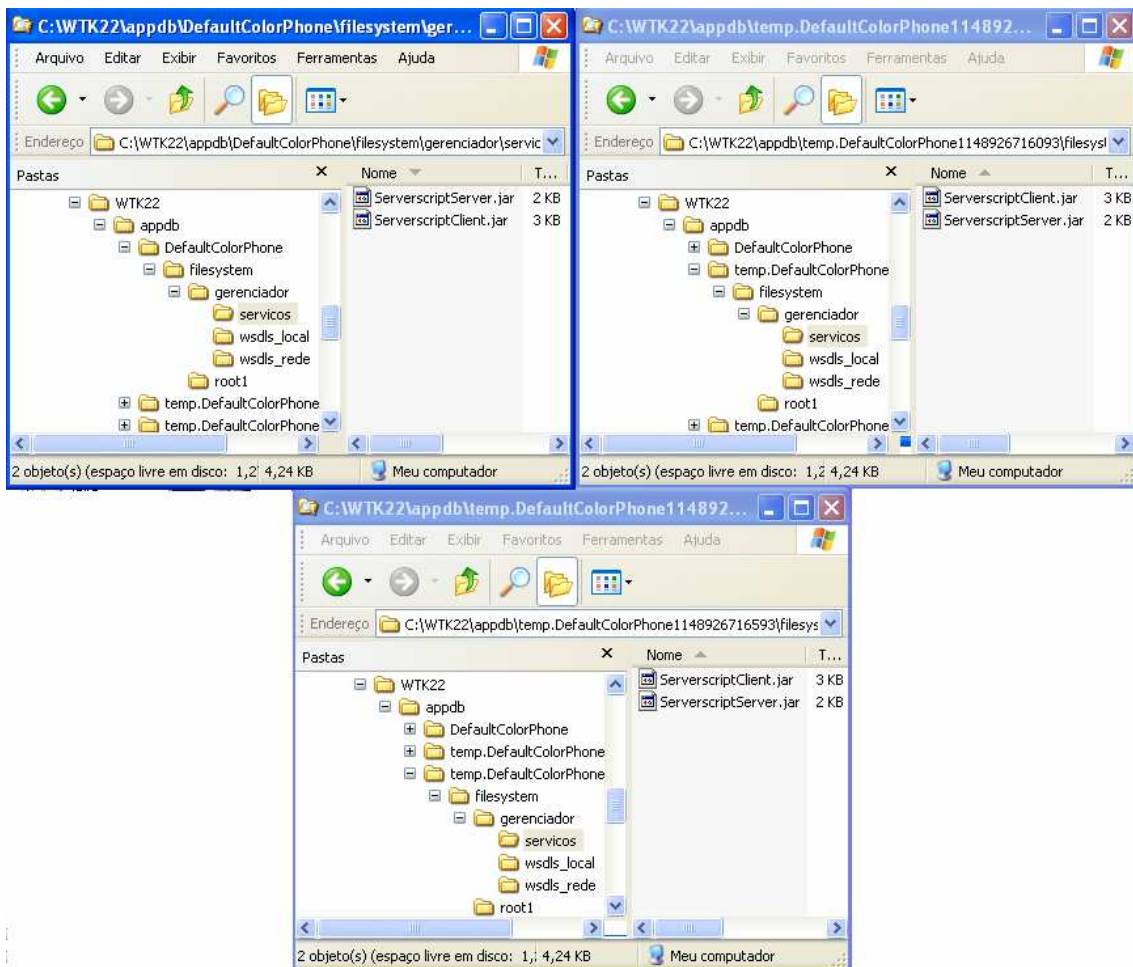


FIGURA 5.11 Conteúdo do diretório serviço de cada dispositivo da rede



Já os testes nos dispositivos móveis reais, não foi possível pela inviabilidade de se conseguir cartões de expansão para os *palms*, tendo em vista que sem o *memory card*, a biblioteca *FileConnection* não funciona.

Desta forma, apenas colocamos em prática os testes com o UDDI que são bem simples:

- Foi armazenado os serviços em um site, como por exemplo:

[www.inf.ufsc.br/~miyakawa/testes](http://www.inf.ufsc.br/~miyakawa/testes).

- Em seguida, publicado os serviços juntamente com o endereço no UDDI@SAP.

- Então simulamos uma pesquisa pelo serviço chamado "Hello" e efetuamos o *download* através da URL obtida na pesquisa, já que o serviço é executado localmente [Faria et al., 2004].

- Tendo os serviços localmente, os mesmos seriam transferidos para os dispositivos móveis reais.

Nas figuras 5.11 e 5.12, são apresentados o resultado da pesquisa pelo serviço chamado "Hello" e o endereço onde o mesmo pode ser localizado.

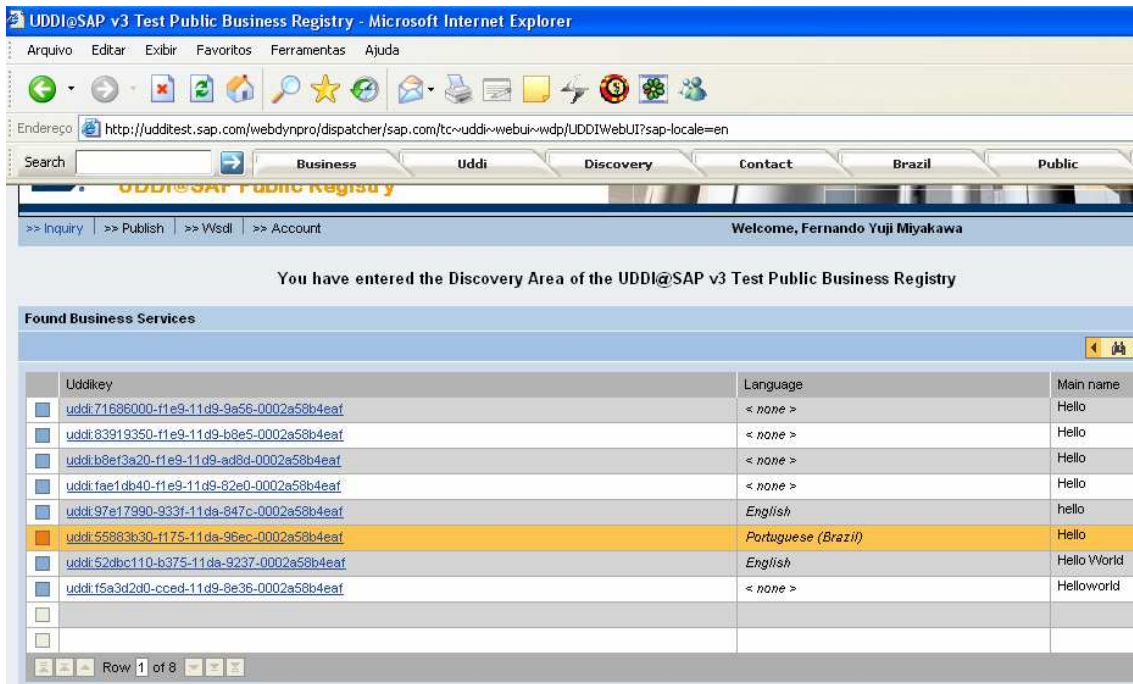


FIGURA 5.12 Resultado da pesquisa

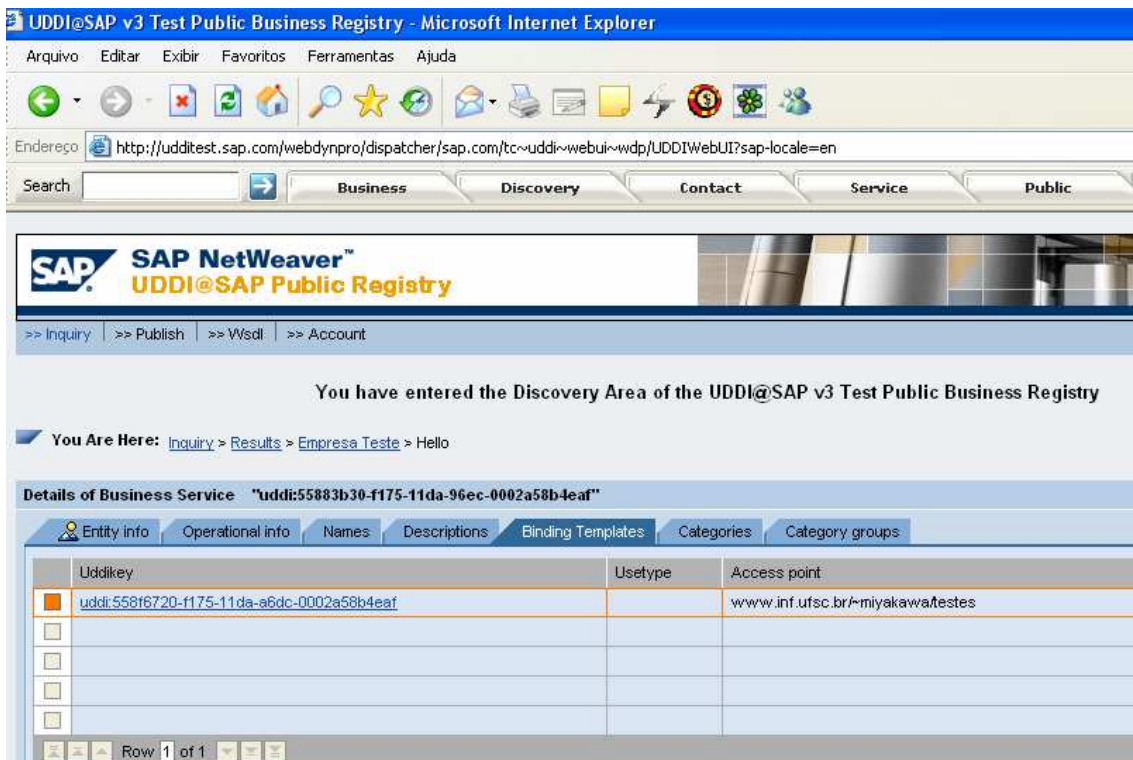


FIGURA 5.13 Endereço do serviço pesquisado

Durante a fase de testes, algumas limitações foram observadas, que podem ser analisadas a seguir:

- A API *JSR75 FileConnection* além de ser suportada por poucos dispositivos atualmente, também apresenta uma dificuldade que não é possível ser criado o diretório *root* em tempo de execução e, sendo assim, foi criado manualmente a cada teste.
- A API *JSR75 FileConnection*, como foi dito anteriormente, só pode ser utilizada em dispositivos móveis com cartões de expansão.
- Com a finalidade de não sobrecarregar o processador do dispositivo, adotou-se processar uma conexão (dentro de uma *thread*) por vez, já que o tempo de processamento de cada conexão é extremamente rápido. Mas, pode efetuar várias ofertas e *downloads* ao mesmo tempo.

Tendo em vista do que foi apresentado, apesar da inviabilidade dos testes em dispositivos reais, os testes no simulador foram bem sucedidos e, assim como foi demonstrado, a utilização da aplicação é uma alternativa interessante para o fornecimento de *web services* a usuários de redes *ad hoc*. A aplicação torna factível levar serviços a lugares inacessíveis a rede fixa, lugares onde a necessidade é temporária, buscando de fato, uma computação disponível a qualquer hora e em qualquer lugar.

## 6 Conclusões e Trabalhos Futuros

Tendo em vista do que foi apresentado, podemos verificar que o modelo seguido realmente possibilita disponibilizar serviços sem a necessidade de infra-estrutura fixa. Desta forma, possibilitando usuários acessar aplicações realmente de modo *Ad hoc*, explorando as características desses ambientes.

Com a implementação dessa aplicação para prover e executar web services em redes Ad Hoc, pode-se concluir que é uma opção interessante e eficiente para ambientes onde não é possível ou não é viável ter acesso a uma rede fixa. Embora tenhamos muitas limitações e uma série de melhorias e recursos adicionais a serem aderidos à aplicação, a adoção da aplicação para o fornecimento de serviços móveis minimiza os diversos desafios encontrados quando desejamos levar serviços aos usuários destes ambientes tornando-se um diferencial dentro da topologia *Ad hoc* por permitir a execução dos serviços sem auxílio de qualquer infra-estrutura fixa.

Como trabalhos futuros, seria interessante a implementação de um mecanismo que possa medir a taxa de transmissão de *downloads* e *uploads* de dados e a largura de banda disponível nas conexões realizadas. Uma outra necessidade seria que a aplicação possa permitir o controle do número máximo de conexões ativas de modo automatizado para que o dispositivo não fique sobrecarregado. Além dessas questões, poderia existir suporte a alta disponibilidade do servidor de serviços da aplicação e segurança das conexões de troca de mensagens.

## Referências Bibliográficas

CHAPPELL, D.; JEWELL, T. **Java Web Services**. O'Reilly, 2002.

DANTAS, Mário. **Tecnologias de redes de comunicação e computadores**.  
Rio de Janeiro: Axcel Books, 2002.

FARIA, F.; MAZZOLA, V.; DANTAS M.A.R. **Servidores Móveis em Redes Ad hoc**. II Escola Regional de Redes de Computadores, 129-134, Canoas - RS. Jul. 2004.

FRIEDMAN, R. **Caching Web Services in Mobile Ad-Hoc Networks: Opportunities and Challenges**, ACM Workshop On Principles Of Mobile Computing, p. 90-96, Oct. 2002.

KSOAP ORG. Technical Report <<http://www.ksoap.org/>>. Disponível on-line, acesso em 16/05/2006.

KXML. Disponível em <<http://kxml.sourceforge.net/>>. Acesso em: 10/12/2005.

MATEUS, G. R.; LOUREIRO, A. A. F. **Introdução a Computação Móvel**. 1.ed. DCC/IM, COPPE/Sistemas, NCE/UFRJ, 11a. Escola de Computação, 1998.

MATEUS, G.R.; LOUREIRO, A. A. F. **Introdução à Computação Móvel**. 2.ed. Disponível em <[http://www.dcc.ufmg.br/~loureiro/cm/docs/cm\\_livro\\_2e.pdf](http://www.dcc.ufmg.br/~loureiro/cm/docs/cm_livro_2e.pdf)>. Acesso em: 10/05/2006.

MUCHOW, J. W. **Core J2ME Technology & MIDP**. Prentice Hall, 2001.

OASIS Member Section, **Universal Description, Discovery e Integration** <<http://uddi.org/pubs/uddi-v3.0.1-20031014.htm>>. Disponível on-line, acesso em 14/11/2005.

PINHEIRO, J. M. S. **Redes Móveis Ad Hoc**. Abr. 2005. Disponível em <[http://www.projetederedes.com.br/artigos/artigo\\_redes\\_moveis\\_ad\\_hoc.php](http://www.projetederedes.com.br/artigos/artigo_redes_moveis_ad_hoc.php)>. Acesso em: 15/05/2006.

SADOK, D. H. F.; D'OLIVEIRA, S. T. J. **Análise de Tráfego de Dados em Redes Bluetooth**. 2001. 51 f. Centro de Informática, Universidade Federal de Pernambuco, Recife, 2001.

STEELE, R. **A Web Services-based System for Ad-hoc Mobile Application Integration**. Coding and Computing ITCC International Conference. IEEE, p. 248-252. Apr. 2003.

SUN Microsystems. Disponível em <<http://www.sun.com/>>. Acesso em: 12/01/2006.

XML ORG. **Extensible Markup Language (XML)**. Technical report  
<<http://www.xml.org/xml/xmldev.shtml>>. Disponível on-line, acesso em  
16/05/2006.

(W3C, A), **Web Services. Technical report**. The World Wide Web Consortium  
<<http://www.w3.org/TR/2004/NOTE-ws-arch-20040211>>. Disponível on-line,  
acesso em 16/05/2006.

(W3C, B), **Web Services Description Language (WSDL). Technical report**.  
The World Wide Web Consortium <<http://www.w3.org/TR/wsdl>>. Disponível on-  
line, acesso em 16/05/2006.

(W3C, C), **Extensible Markup Language (XML). Technical report**. The World  
Wide Web Consortium <<http://www.w3.org/TR/2004/REC-xml-20040204/>>.  
Disponível on-line, acesso em 16/05/2006.

(W3C, D), **Document Object Model (DOM). Technical report**. The World  
Wide Web Consortium <<http://www.w3.org/DOM/>>. Disponível on-line, acesso  
em 16/05/2006.

(W3C, E), **XML Schema. Technical report**. The World Wide Web Consortium  
<<http://www.w3.org/XML/Schema>>. Disponível on-line, acesso em 16/05/2006.

(W3C, F), **Extensible Stylesheet Language (XSL). Technical report.** The World Wide Web Consortium <<http://www.w3.org/Style/XSL/>>. Disponível on-line, acesso em 16/05/2006.

(W3C, G), **XML Pointer, XML Base and XML Linking. Technical report.** The World Wide Web Consortium <<http://www.w3c.org/XML/Linking>>. Disponível on-line, acesso em 16/05/2006.

(W3C, I), **Simple Object Access Protocol (SOAP). Technical report.** The World Wide Web Consortium <<http://www.w3c.org/TR/2003/REC-soap12-part0-20030624>>. Disponível on-line, acesso em 16/05/2006.

(W3C, J), **XML Information Set. Technical report.** The World Wide Web Consortium <<http://www.w3.org/TR/xml-infoset/>>. Disponível on-line, acesso em 16/05/2006.

WANG, H; ZHANG, Y.; CAO, J.; et al. **Achieving secure and flexible M-services through tickets.** IEEE Transactions on Systems, Man and Cybernetics, Part A, Volume: 33, Issue: 6, p. 697-708, Nov. 2003.

WOMBACHER, A.; MAHLEKO, B.; **Ad-hoc business processes in web services.** Symposium on Applications and the Internet Workshops, 2003. Proceedings. 2003, 27-31 Jan. 2003.



YANG, X.; BOUGUETTAYA, A.; MEDJAHED, B.; et al. **Organizing and Accessing Web Services on Air**, Systems, Man and Cybernetics, IEEE Transactions, p. 742-757, Nov. 2003.

YUAN, M. J.; LONG, J. **Java Readies Itself for Wireless Web Services.**

<<http://www.javaworld.com/javaworld/jw-06-2002/jw-0621-wireless.html>>, 2002.

Disponível on-line, acesso em 18/03/2006.

# Apêndices

## Classe GerenciadorMIDlet

```
package gerenciador;

import java.util.Vector;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

import comunicacao.CarregadorArquivo;
import comunicacao.GerenteSocket;
import comunicacao.ServidorSocket;

public class GerenciadorMIDlet extends MIDlet
    implements CommandListener, ItemCommandListener{

    protected GerenteSocket servidor;
    private Vector configuracoesSockets;

    protected ServidorSocket teste;

    protected Display display;
    protected Form formPrincipal;
    protected Form fAux;

    private ChoiceGroup cgServicosRede;
    private ChoiceGroup cgServicosLocais;

    private String[] nomesServicosRede = {"somador", "multiplicador"};
    private String[] nomesServicosLocais;

    private final static Command CMD_SAIR =
        new Command("Sair", Command.EXIT, 1);
    private final static Command CMD_VOLTAR =
        new Command("Voltar", Command.BACK, 1);
    private final static Command CMD_INFO_REDE =
        new Command("Info", Command.ITEM, 1);
    private final static Command CMD_INFO_LOCAL =
        new Command("Info", Command.ITEM, 1);
    private final static Command CMD_DOWNLOAD =
        new Command("Download", Command.ITEM, 1);
    private final static Command CMD_OFERECER =
        new Command("Oferecer", Command.ITEM, 1);

    public void initializeComponentes(){

        formPrincipal = new Form("Gerenciador");
        fAux = new Form ("Gerenciador");

        StringItem item = new StringItem("Nós ativos no momento: ", "0");
        formPrincipal.append(item);
```

```

        inicializeServidor();

        cgServicosRede = new ChoiceGroup("Serviços disponíveis na rede: ",
            Choice.EXCLUSIVE, nomesServicosRede, null);
        formPrincipal.append(cgServicosRede);

        StringItem btDownload = new StringItem("Download", "", Item.BUTTON);
        btDownload.addCommand(CMD_DOWNLOAD);
        btDownload.setItemCommandListener(this);
        formPrincipal.append(btDownload);

        StringItem btInfo = new StringItem("+ Info", "", Item.BUTTON);
        btInfo.setDefaultCommand(CMD_INFO_REDE);
        btInfo.setItemCommandListener(this);
        formPrincipal.append(btInfo);

        inicializeServicos();

        cgServicosLocais = new ChoiceGroup("Serviços locais: ",
            Choice.EXCLUSIVE, nomesServicosLocais, null);
        formPrincipal.append(cgServicosLocais);

        StringItem btOferecerServico = new StringItem("Oferecer Serviço", "",
            Item.BUTTON);
        btOferecerServico.setDefaultCommand(CMD_OFERECER);
        btOferecerServico.setItemCommandListener(this);
        formPrincipal.append(btOferecerServico);

        btInfo = new StringItem("+ Info", "", Item.BUTTON);
        btInfo.setDefaultCommand(CMD_INFO_LOCAL);
        btInfo.setItemCommandListener(this);
        formPrincipal.append(btInfo);

        formPrincipal.addCommand(CMD_SAIR);
        formPrincipal.setCommandListener(this);

        display = Display.getDisplay(this);
        display.setCurrent(formPrincipal);
    }

    private void inicializeServidor(){

        CarregadorArquivo c = new CarregadorArquivo("leitord_configuracao");
        c.start();
        configuracoesSockets = c.getDadosConfiguracoes();

        servidor = new GerenteSocket(this, true, "",
            (String)configuracoesSockets.elementAt(1), false);
        servidor.start();
    }

    private void inicializeServicos(){

        CarregadorArquivo c = new CarregadorArquivo(
            "leitord_wsdl_local", "service", "name", new Vector());
        c.start();

        Vector nomesServicos = c.getDadosWsdl();
        nomesServicosLocais= new String[nomesServicos.size()];
        nomesServicos.copyInto(nomesServicosLocais);
    }

```

```

    }

    protected void startApp() {
        inicializeComponentes();
    }

    public void commandAction(Command c, Displayable d) {
        if (c == CMD_VOLTAR){
            volte();
        }else if (c == CMD_SAIR){
            saia();
        }
    }
}

private void volte(){
    display.setCurrent(formPrincipal);
}

private void saia(){
    destroyApp(true);
    notifyDestroyed();
}

public void commandAction(Command c, Item item) {
    if (c == CMD_INFO_REDE){
        getInfoServicoRede();
    }else if (c == CMD_INFO_LOCAL){
        getInfoServicoLocal();
    }else if (c == CMD_DOWNLOAD){
        download();
    }else if (c == CMD_OFERECER){
        oferecaServico();
    }
}

private void getInfoServicoRede(){

    int iSelecionado = cgServicosRede.getSelectedIndex();
    String nomeServico = nomesServicosRede[iSelecionado];

    fAux.deleteAll();
    fAux.append(nomeServico);
    fAux.addCommand(CMD_VOLTAR);
    fAux.setCommandListener(this);

    display.setCurrent(fAux);
}

private void getInfoServicoLocal(){

    int iSelecionado = cgServicosLocais.getSelectedIndex();
    String nomeServico = nomesServicosLocais[iSelecionado];

    fAux.deleteAll();
    fAux.append(nomeServico);
    fAux.addCommand(CMD_VOLTAR);
    fAux.setCommandListener(this);

    display.setCurrent(fAux);
}
}

```

```

private void download(){

    int iSelecionado = cgServicosRede.getSelectedIndex();
    String nomeServico = nomesServicosRede[iSelecionado];

    GerenteSocket umCliente = new GerenteSocket(this, false,
                                                (String)configuracoesSockets.elementAt(0),
                                                (String)configuracoesSockets.elementAt(2),
                                                true);
    umCliente.setDadosCliente(nomeServico.getBytes());
    umCliente.setNomeServico(nomeServico);
    umCliente.start();

    fAux.deleteAll();
    fAux.append(nomeServico);

    fAux.addCommand(CMD_VOLTAR);
    fAux.setCommandListener(this);
    display.setCurrent(fAux);
}

private void oferecaServico(){

    int iSelecionado = cgServicosLocais.getSelectedIndex();
    String nomeServico = nomesServicosLocais[iSelecionado];

    envieBroadcast(nomeServico);

    fAux.deleteAll();
    fAux.append(nomeServico);
    fAux.addCommand(CMD_VOLTAR);
    fAux.setCommandListener(this);
    display.setCurrent(fAux);
}

private void envieBroadcast(String nomeServico){

    CarregadorArquivo carregador = new CarregadorArquivo("leitor", null, 0);
    carregador.setNomeArquivo("wsdls_local/" + nomeServico + ".wsdl");
    carregador.start();

    for(int j=0; j<configuracoesSockets.size()-3; j++){
        GerenteSocket umCliente = new GerenteSocket(this, false,
                                                    (String)configuracoesSockets.elementAt(0),
                                                    (String)configuracoesSockets.elementAt(j+3),
                                                    false);
        umCliente.envieCliente(carregador.getArquivo());
        umCliente.start();
    }
}

public void atualizeServicosRede(){

    Vector nomesNovos = new Vector();
    for(int i=0; i<nomesServicosRede.length; i++){
        nomesNovos.addElement(nomesServicosRede[i]);
    }
    CarregadorArquivo c = new CarregadorArquivo("leitor_wsdls_rede",
        "service", "name", nomesNovos);
}

```

```

        c.start();

        Vector wsdlNovos = (Vector)c.getDadosWsd();
        for(int i=0; i<wsdlNovos.size(); i++){
            nomesNovos.addElement((String) wsdlNovos.elementAt(i));
        }
        nomesServicosRede = new String[nomesNovos.size()];
        nomesNovos.copyInto(nomesServicosRede);

        cgServicosRede = new ChoiceGroup("Serviços disponíveis na rede: ",
            Choice.EXCLUSIVE, nomesServicosRede, null);

        formPrincipal.set(1, cgServicosRede);
        display.setCurrent(formPrincipal);
    }

    public void atualizeServicosLocal(){

        Vector nomesNovos = new Vector();
        for(int i=0; i<nomesServicosLocais.length; i++){
            nomesNovos.addElement(nomesServicosLocais[i]);
        }
        CarregadorArquivo c = new CarregadorArquivo("leitord_wsdls_local",
            "service", "name", nomesNovos);
        c.start();

        Vector wsdlNovos = (Vector)c.getDadosWsd();
        for(int i=0; i<wsdlNovos.size(); i++){
            nomesNovos.addElement((String) wsdlNovos.elementAt(i));
        }
        nomesServicosLocais = new String[nomesNovos.size()];
        nomesNovos.copyInto(nomesServicosLocais);

        cgServicosLocais = new ChoiceGroup("Serviços locais: ",
            Choice.EXCLUSIVE, nomesServicosLocais, null);

        formPrincipal.set(4, cgServicosLocais);
        display.setCurrent(formPrincipal);
    }

    public void destroyApp(boolean unconditional) {
        if (servidor != null){
            servidor.fecheConexao();
        }
    }

    protected void pauseApp() {
    }
}

```

# Classe GerenteSocket

```
package comunicacao;

import comunicacao.ServidorSocket;
import comunicacao.ClienteSocket;

import gerenciador.GerenciadorMIDlet;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.util.Vector;

public class GerenteSocket implements Runnable{

    protected GerenciadorMIDlet aInterface;
    protected CarregadorArquivo carregadorArquivo;
    protected EncapsulamentoSOAP encapsulador;

    protected ServerRunner servidor;
    protected ServidorSocket servidorSocket;
    protected ClienteSocket clienteSocket;

    protected DataInputStream entrada;
    protected DataOutputStream saida;

    protected String ip;
    protected String porta;
    protected byte[] dados;

    private Vector filaConexoes;
    private boolean ehServidor;
    private String nomeServico;

    private final int SINAL_DOWNLOAD = -3;
    private String NOME_ARQUIVO_PADRAO =
        "wsdls_rede/descricaoServico.wsdl";

    public GerenteSocket(GerenciadorMIDlet interfaces ,boolean tipo, String ip,
        String porta, boolean escreve_e_le){
        this.aInterface = interfaces;
        this.ehServidor = tipo;
        this.ip = ip;
        this.porta = porta;
        this.filaConexoes = new Vector();
        if (ehServidor){
            inicializeServidor(porta);
        }else{
            inicializeCliente(ip, porta, escreve_e_le);
        }
    }

    public void start(){
        Thread t = new Thread(this);
        t.start();
    }
}
```

```

public void inicializeServidor(String porta){
    servidor = new ServerRunner(this, porta);
    servidorSocket = null;
}

public void inicializeCliente(String ip, String porta, boolean escreve_e_le){
    clienteSocket = new ClienteSocket(this, ip, porta, escreve_e_le);
}

public void run(){
    if(ehServidor){
        servidor.start();

        while(true){
            servidorSocket = getConexao();
            servidorSocket.start();

            dados = getDadosServidor();
            int sinal = getSinalServidor();
            int tamanho = getTamanhoDadosServidor();

            if (sinal != SINAL_DOWNLOAD){

                encapsulador = new EncapsulamentoSOAP(dados,null);
                dados = (byte[])
                    encapsulador.retireEncapsulamento().elementAt(0);
                tamanho = dados.length;

                CarregadorArquivo c = crieCarregadorArquivo("escritor",
                    dados, tamanho, NOME_ARQUIVO_PADRAO);
                System.out.println("Vai escrever: " +
                    NOME_ARQUIVO_PADRAO );
                atualizeServicosRede(c);
            }else{

                nomeServico = new String(dados, 0, tamanho);
                CarregadorArquivo c = crieCarregadorArquivo("leitor", null,
                    0, "servicos/" + nomeServico + "Server.jar");
                byte[] servicoServidor = c.getArquivo();

                c = crieCarregadorArquivo("leitor", null, 0, "servicos/" +
                    nomeServico + "Client.jar");
                byte[] servicoCliente = c.getArquivo();

                encapsulador = new EncapsulamentoSOAP(servicoServidor,
                    servicoCliente);
                byte[] dadosEncapsulados =
                    encapsulador.efetueEncapsulamento();

                envieServidor(dadosEncapsulados);
                nomeServico="";
            }
            elimineConexao();
        }
    }else{
        clienteSocket.start();

        if(clienteSocket.isEscritor_leitor()){

```



```

        byte[] dadosRecebidos = getDadosCliente();

        encapsulador = new EncapsulamentoSOAP(dadosRecebidos, null);
        Vector servicos = encapsulador.retireEncapsulamento();

        byte[] servicoServidor = (byte[]) servicos.elementAt(0);
        crieCarregadorArquivo("escritor",
                               servicoServidor,
                               servicoServidor.length,
                               "servicos/" + nomeServico + "Server.jar");

        byte[] servicoCliente = (byte[]) servicos.elementAt(1);
        crieCarregadorArquivo("escritor",
                               servicoCliente,
                               servicoCliente.length,
                               "servicos/" + nomeServico + "Client.jar");

        CarregadorArquivo c = crieCarregadorArquivo("copiador",
                                                    null,
                                                    0,
                                                    "wsdls_rede/" + nomeServico + ".wsdl");
        atualizeServicosLocal(c);
    }
}
fecheConexao();
}

private synchronized ServidorSocket getConexao(){
    try{
        if (filaConexoes.isEmpty())
            wait();
    }catch(Exception e){
        e.printStackTrace();
    }
    return (ServidorSocket) filaConexoes.elementAt(0);
}

private CarregadorArquivo crieCarregadorArquivo(String tipo, byte[] umArquivo ,
                                                int tamArquivo, String nomeArquivo){

    CarregadorArquivo carregador = new CarregadorArquivo(tipo,
                                                         umArquivo, tamArquivo);
    carregador.setNomeArquivo(nomeArquivo);
    carregador.start();
    return carregador;
}

public synchronized void envieCliente(byte[] d){

    this.dados = d;
    encapsulador = new EncapsulamentoSOAP(dados, null);
    dados = encapsulador.efetueEncapsulamento();

    clienteSocket.setDados(dados);
    clienteSocket.carregouDados();
}

public synchronized void envieServidor(byte[] d){
    this.dados = d;

```

```

        servidorSocket.setDados(dados);
        servidorSocket.carregouDados();
    }

    public synchronized byte[] getDadosServidor(){
        try{
            wait();
            this.dados = servidorSocket.getDados();
        }catch (Exception e) {
            e.printStackTrace();
        }
        return dados;
    }

    public synchronized byte[] getDadosCliente(){
        try{
            wait();
            this.dados = clienteSocket.getDados();
        }catch (Exception e) {
            e.printStackTrace();
        }
        return dados;
    }

    public synchronized void chegouDados(){
        try{
            notify();
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void atualizeServicosLocal(CarregadorArquivo c) {
        try{
            c.getArquivo();
            aInterface.atualizeServicosLocal();
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void atualizeServicosRede(CarregadorArquivo c) {
        try{
            c.getArquivo();
            aInterface.atualizeServicosRede();
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void fecheConexao(){
        if (ehServidor){
            if (servidor != null)
                servidor.pare();
            if (servidorSocket != null)
                servidorSocket.fecheConexao();
        }else
            clienteSocket.fecheConexao();
    }
}

```

```

public void setDadosCliente(byte[] d){
    clienteSocket.setDados(d);
}

private int getTamanhoDadosCliente(){
    return clienteSocket.getTamanhoDados();
}

private int getTamanhoDadosServidor(){
    return servidorSocket.getTamanhoDados();
}

private int getSinalServidor(){
    return servidorSocket.getSinal();
}

public void setNomeServico(String nomeServico) {
    this.nomeServico = nomeServico;
}

public synchronized void adicioneConexao(ServidorSocket ss){
    this.filaConexoes.addElement(ss);
    if (filaConexoes.size() == 1)
        notify();
}

private void elimineConexao(){
    this.filaConexoes.removeElementAt(0);
}
}

```

## Classe ServidorSocket

```

package comunicacao;

import javax.microedition.io.*;
import java.io.*;

public class ServidorSocket extends Thread{

    protected GerenteSocket gerente;

    protected SocketConnection conexao;
    protected DataInputStream entrada;
    protected DataOutputStream saida;

    protected String porta;
    protected byte[] dados;
    protected int tamanhoDados;

    private int sinal;
    private boolean fechou;

    private final int SINAL_DOWNLOAD = -3;

```

```

public ServidorSocket(GerenteSocket umGerente,
                     SocketConnection umaConexao){
    this.gerente = umGerente;
    this.conexao = umaConexao;
    this.dados = null;
    this.tamanhoDados = 0;
    fechou = false;
}

public void abraStreams(){
    try{
        fechou = false;
        saida = conexao.openDataOutputStream();
        entrada = conexao.openDataInputStream();
    }catch(Exception e){
        if(!fechou)
            e.printStackTrace();
    }
}

public void run(){
    try{
        abraStreams();
        if (!fechou){
            leiaDados();
            sinal = entrada.readInt();
            gerente.chegouDados();

            System.out.println("sinal: "+sinal);
            if(sinal == SINAL_DOWNLOAD){
                espereDados();
                escrevaDados();
            }
            sleep(1000);
        }
        fecheConexao();
    }catch(Exception e){
        e.printStackTrace();
    }
}

private void leiaDados(){
    try{
        int c = 0;

        int i = 0;

        dados = new byte[30000];

        while (((c = entrada.read()) != -1) && ((int)c != 4)){
            dados[i++] = (byte) c;
        }
        tamanhoDados = i;
    }catch(Exception e){
        e.printStackTrace();
    }
}

private void escrevaDados(){
    try{
        saida.write(dados);
        saida.flush();
        System.out.println("enviou: " + new String(dados));
    }catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

    }

    public synchronized void espereDados() {
        try{
            wait();
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    public synchronized void carregouDados() {
        try{
            notify();
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    public void fecheConexao(){
        try{
            fechou=true;
            if (entrada != null) {
                entrada.close();
            }
            if (saida != null) {
                saida.close();
            }
            if (conexao != null) {
                conexao.close();
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    public DataInputStream getEntrada() {
        return entrada;
    }

    public DataOutputStream getSaida() {
        return saida;
    }

    public void setDados(byte[] dados) {
        this.dados = dados;
    }

    public byte[] getDados(){
        return dados;
    }

    public int getTamanhoDados() {
        return tamanhoDados;
    }

    public int getSinal() {
        return sinal;
    }
}

```

# Classe ClienteSocket

```
package comunicacao;

import java.io.DataInputStream;
import java.io.DataOutputStream;

import javax.microedition.io.Connector;
import javax.microedition.io.SocketConnection;

public class ClienteSocket extends Thread{

    protected SocketConnection conexao;
    protected GerenteSocket gerente;

    protected DataInputStream entrada;
    protected DataOutputStream saida;

    protected String ip;
    protected String porta;
    protected byte[] dados;
    protected int tamanhoDados;

    private boolean escritor_leitor;
    private final int SINAL_DOWNLOAD = -3;
    private final int SINAL_OFERTA_SERVICO = -4;

    public ClienteSocket(GerenteSocket g, String host,
                        String porta, boolean le_e_escreve){
        this.gerente = g;
        this.ip = host;
        this.porta = porta;
        this.escritor_leitor = le_e_escreve;
    }

    public void abraConexao(){
        try{
            conexao = (SocketConnection)
                Connector.open("socket://" + ip + ":" + porta);
            System.out.println(
                "Cliente conectou-se em " + "socket://" + ip + ":" + porta);
            saida = conexao.openDataOutputStream();
            entrada = conexao.openDataInputStream();
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    public void run(){
        abraConexao();

        try {
            if (dados != null){
                escrevaDados();
                if(escritor_leitor){
                    sinalize(SINAL_DOWNLOAD);
                    leiaDados();
                    gerente.chegouDados();
                }
            }
        }
    }
}
```

```

        }else{
            sinalize(SINAL_OFERTA_SERVICO);
        }
        sleep(1000);
    }else{
        System.err.println("Nao carregou dados no Cliente (dados=null!!");
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
fecheConexao();
}

private void leiaDados(){
    try{
        int c = 0;
        int i = 0;

        dados = new byte[30000];

        while ((c = entrada.read()) != -1) {
            dados[i++] = (byte) c;
        }
        tamanhoDados = i;
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void escrevaDados(){
    try{
        saida.write(dados);
        saida.write(4);
        System.out.println("escreveu :" + new String(dados));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void sinalize(int sinal){
    try{
        saida.writeInt(sinal);
        saida.flush();
    } catch (Exception e){
        e.printStackTrace();
    }
}

public synchronized void carregouDados() {
    try{
        notify();
    } catch (Exception e){
        e.printStackTrace();
    }
}

public void fecheConexao(){
    try{
        if (entrada != null) {
            entrada.close();
        }
    }
}

```

```

        }
        if (saida != null) {
            saida.close();
        }
        if (conexao != null) {
            conexao.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public DataInputStream getEntrada() {
    return entrada;
}

public DataOutputStream getSaida() {
    return saida;
}

public void setDados(byte[] dados) {
    this.dados = dados;
}

public byte[] getDados() {
    return dados;
}

public int getTamanhoDados() {
    return tamanhoDados;
}

public boolean isEsritor_leitor() {
    return escritor_leitor;
}
}

```

## Classe ServerRunner

```

package comunicacao;

import javax.microedition.io.Connector;
import javax.microedition.io.ServerSocketConnection;
import javax.microedition.io.SocketConnection;

public class ServerRunner extends Thread {

    protected GerenteSocket gerente;
    protected ServerSocketConnection conexaoServidora;
    protected SocketConnection conexao;
    protected String porta;

    private boolean parou;
}

```



```

public ServerRunner(GerenteSocket umGerente, String porta){
    this.gerente = umGerente;
    this.porta = porta;
    this.parou = false;
    try{
        conexaoServidora = (ServerSocketConnection)
            Connector.open("socket://:" + porta);
    }catch(Exception e){
        e.printStackTrace();
    }
}

private void abraConexao(){
    try{
        parou = false;
        System.out.println("Aguardando clientes...");
        conexao = (SocketConnection)
            conexaoServidora.acceptAndOpen();
        System.out.println("Um cliente conectado!");
    }catch(Exception e){
        if(!parou)
            e.printStackTrace();
    }
}

public void run(){
    while(true){
        abraConexao();
        if(!parou){
            ServidorSocket thServidor =
                new ServidorSocket(gerente, conexao);
            gerente.adicioneConexao(thServidor);
            conexao=null;
        }else
            break;
    }
}

public void pare(){
    try{
        parou=true;
        if (conexaoServidora != null)
            conexaoServidora.close();
        if (conexao != null)
            conexao.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}
}

```

## Classe EncapsulamentoSOAP

```

package comunicacao;

import java.io.ByteArrayInputStream;

```

```

import java.io.ByteArrayOutputStream;
import java.util.Vector;

import org.ksoap2.*;
import org.ksoap2.serialization.*;

import org.kxml2.io.*;
import org.xmlpull.v1.*;

public class EncapsulamentoSOAP {

    protected byte[] dados1;
    protected byte[] dados2;

    public EncapsulamentoSOAP(byte[] d1, byte[] d2){
        dados1 = d1;
        dados2 = d2;
    }

    public byte[] efetueEncapsulamento(){

        byte[] encapsulados = new byte[1];
        try {

            SoapSerializationEnvelope envelope =
                new SoapSerializationEnvelope(SoapEnvelope.VER12);
            SoapObject objetoSoap = new SoapObject("", "getObjeto");
            objetoSoap.addProperty("dados1",
                new String(dados1,"ISO-8859-1"));
            if (dados2 != null)
                objetoSoap.addProperty("dados2",
                    new String(dados2,"ISO-8859-1"));
            envelope.bodyOut = objetoSoap;
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            XmlSerializer xw = new KXmlSerializer();
            xw.setOutput(bos, "ISO-8859-1");

            envelope.encodingStyle = "ISO-8859-1";
            envelope.write(xw);
            xw.flush();
            bos.write('\r');
            bos.write('\n');
            encapsulados = bos.toByteArray();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return encapsulados;
    }

    public Vector retireEncapsulamento(){

        byte[] dadosOriginais1 = new byte[0];
        byte[] dadosOriginais2 = new byte[0];
        Vector dados = new Vector();
        ByteArrayInputStream entrada;
        try {
            SoapSerializationEnvelope envelope =

```

```

        new SoapSerializationEnvelope(
            SoapSerializationEnvelope.VER12);

        entrada = new ByteArrayInputStream(dados1);
        XmlPullParser parser = new KXmlParser();
        parser.setInput(entrada, "ISO-8859-1");
        parser.setFeature(
            XmlPullParser.FEATURE_PROCESS_NAMESPACES, true);
        envelope.parse(parser);
        SoapObject objetoSOAP = (SoapObject) envelope.bodyIn;

        dados.addElement(dadosOriginais1);
        dados.addElement(dadosOriginais2);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return dados;
}
}

```

## Classe CarregadorArquivo

```

package comunicacao;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.util.Enumeration;
import java.util.Vector;

import javax.microedition.io.Connector;
import javax.microedition.io.file.*;

public class CarregadorArquivo implements Runnable{

    protected byte[] arquivo;
    protected WsdParser parser;

    protected String tipo;
    protected String nomeArquivo;
    protected Vector dadosWsd;
    protected Vector dadosConfiguracoes;

    private int tamanhoArquivo;
    private Vector wsdlExistentes;
    private boolean liberouDados;

    private final String ESCRITOR = "escritor";
    private final String LEITOR = "leitor";
    private final String COPIADOR = "copiador";
    private final String LEITOR_WSDLS_LOCAL = "leitor_wsdl_local";
    private final String LEITOR_WSDLS_REDE = "leitor_wsdl_rede";
    private final String LEITOR_CONFIGURACAO = "leitor_configuracao";
}

```

```

private final String WSDLS_LOCAL_PATH =
    "file:///gerenciador/wsdls_local/";
private final String WSDLS_REDE_PATH = "file:///gerenciador/wsdls_rede/";
private final String ROOT_PATH = "file:///gerenciador/";
private final String ARQUIVO_CONFIGURACAO = "conf.txt";
private final String WSDLS_LOCAL_DIR = "wsdls_local";
private final String WSDLS_REDE_DIR = "wsdls_rede";
private final String SERVICOS_DIR = "servicos";

public CarregadorArquivo(String umTipo, String namespace,
    String nomeAtributo, Vector wsdlsExistentes){
    this.tipo = umTipo;
    this.parser = new WsdlParser(namespace, nomeAtributo);
    this.dadosWsdL = new Vector();
    this.wsdlsExistentes = wsdlsExistentes;
    this.liberouDados = false;
}

public CarregadorArquivo(String umTipo, byte[] umArquivo, int umTamanho){
    this.tipo = umTipo;
    this.arquivo = umArquivo;
    this.tamanhoArquivo = umTamanho;
    this.liberouDados = false;
}

public CarregadorArquivo(String umTipo){
    this.tipo = umTipo;
    this.dadosConfiguracoes = new Vector();
    this.liberouDados = false;
}

public void start(){
    Thread t = new Thread(this);
    t.start();
}

public void run(){
    if(tipo.equals(ESCRITOR)){
        crieArquivo();
        carregado();
        System.out.println("escreveu arquivo..");
    }else if(tipo.equals(LEITOR)){
        leiaArquivo();
        System.out.println("leu arquivo...");
    }else if(tipo.equals(COPIADOR)){
        copie_e_cole();
        System.out.println("copiou arquivo para local, agora eh dmf...");
    }else if(tipo.equals(LEITOR_WSDLS_LOCAL)){
        leiaTodosWsdls(WSDLS_LOCAL_DIR);
        System.out.println("leu novos wsdls locais...");
    }else if(tipo.equals(LEITOR_WSDLS_REDE)){
        leiaTodosWsdls(WSDLS_REDE_DIR);
        System.out.println("leu novos wsdls rede...");
    }else if(tipo.equals(LEITOR_CONFIGURACAO)){
        leiaArquivoConfiguracao();
        System.out.println("leu arquivo de configuracoes...");
    }else{
        System.out.println("naoo entro em nada");
    }
    liberouDados = true;
}

```

```

}

public void crieArquivo(){

    if(ehWsdL()){
        verifiqueDiretorio(WSDLS_LOCAL_DIR);
        verifiqueDiretorio(WSDLS_REDE_DIR);
    }else
        verifiqueDiretorio(SERVICOS_DIR);
    try {
        FileConnection fc = (FileConnection)
            Connector.open(ROOT_PATH + nomeArquivo);
        if(!fc.exists()){
            fc.create();
            DataOutputStream os = fc.openDataOutputStream();
            os.write(arquivo, 0, tamanhoArquivo);
            os.close();
        }else{
            System.out.println(
                "Arquivo jah existe, nao serah sobrescrito! " +
                nomeArquivo);
        }
        fc.close();
    }catch(Exception e) {
        e.printStackTrace();
    }
}

public void verifiqueDiretorio(String nomeDiretorio){
    try {
        FileConnection fc = (FileConnection)
            Connector.open(ROOT_PATH + nomeDiretorio + "/");
        if (!fc.exists())
            fc.mkdir();
    }catch(Exception e){
        e.printStackTrace();
    }
}

public synchronized void leiaArquivo(){

    try {
        FileConnection fc = (FileConnection)
            Connector.open(ROOT_PATH + nomeArquivo);

        if(fc.exists()) {
            tamanhoArquivo = (int)fc.fileSize();
            arquivo = new byte[tamanhoArquivo];
            DataInputStream is = fc.openDataInputStream();
            is.read(arquivo, 0, tamanhoArquivo);
        }else
            System.err.println(
                "Nao leu. Arquivo nao existe: " + ROOT_PATH +
                nomeArquivo + " !!");

        carregado();
        fc.close();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

```

```

}
}

public void copie_e_cole(){
    leiaArquivo();
    int nChars = (WSDLS_REDE_DIR+"/").length();
    String nomeArquivoReal = nomeArquivo.substring(
        nChars, nomeArquivo.length());
    System.out.println("nomeArquivoreal: " +nomeArquivoReal);
    setNomeArquivo(WSDLS_LOCAL_DIR + "/" + nomeArquivoReal);
    crieArquivo();
}

public synchronized void leiaTodosWsdl(String nomeDiretorio){

    String diretorioPath;

    if(nomeDiretorio.equals(WSDLS_LOCAL_DIR)){
        diretorioPath = WSDLS_LOCAL_PATH;
    }else{
        diretorioPath = WSDLS_REDE_PATH;
    }
    verifiqueDiretorio(nomeDiretorio);
    try{
        FileConnection fc = (FileConnection)
            Connector.open(diretorioPath);
        if (fc.exists()){
            Enumeration lArquivos = fc.list("*", true);
            fc.close();
            while(lArquivos.hasMoreElements()) {
                nomeArquivo = (String) lArquivos.nextElement();
                if (!wsdlsExistentes.contains(nomeArquivo)){
                    leiaWsd(diretorioPath);
                }
            }
        }else{
            System.err.println("Nao leu. Diretorio nao existe: "
                + diretorioPath + " !!");
        }
        carregado();
    }catch(Exception e){
        e.printStackTrace();
    }
}

public void leiaWsd(String dirPath){
    try{
        FileConnection fc = (FileConnection)
            Connector.open(dirPath + nomeArquivo);
        if (fc.exists()){
            parser.setEntrada(fc.openInputStream());
            String nomeServico = parser.getValorAtributo();
            dadosWsd.addElement( nomeServico );
            if (!nomeArquivo.equals(nomeServico + ".wsdl")){
                if(!lexisteArquivo(
                    dirPath + nomeServico + ".wsdl"))
                    fc.rename( nomeServico + ".wsdl");
                else{
                    fc.delete();
                    System.out.println(

```

```

                "Usuário jah tem wsdl em: " +
                dirPath );
            }
        }else{
            System.out.println("Nao leu. Arquivo nao existe: " +
                dirPath + nomeArquivo + " !!");
        }
        fc.close();
        System.out.println("leu wsdl" );
    }catch(Exception e){
        e.printStackTrace();
    }
}

private boolean existeArquivo(String path){
    boolean existe = false;
    try{
        FileConnection fc = (FileConnection)
            Connector.open(path);
        if (fc.exists()){
            existe = true;
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return existe;
}

public synchronized void leiaArquivoConfiguracao(){
    try{
        FileConnection fc = (FileConnection)
            Connector.open(ROOT_PATH +
                ARQUIVO_CONFIGURACAO);
        if (fc.exists()){
            DataInputStream is = fc.openDataInputStream();
            String palavra = "a";
            while(!palavra.equals("")){
                int c=0;
                palavra = "";
                while(((c = is.read()) != 13) && (c != -1)){
                    palavra += (char)c;
                }
                is.skip(1); // pula o byte '10' -> inicio de linha
                if (!palavra.equals(""))
                    dadosConfiguracoes.addElement(palavra);
                System.out.println("1");
            }
            System.out.println("saiu");
        }else{
            System.out.println("Nao leu. Arquivo nao existe: " +
                ROOT_PATH + ARQUIVO_CONFIGURACAO
                + " !!");
        }
        carregado();
        fc.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}

```

```

    }

    public synchronized byte[] getArquivo() {
        try{
            if (!liberouDados)
                wait();
        }catch(Exception e){
            e.printStackTrace();
        }
        return arquivo;
    }

    public synchronized Vector getDadosWsdI() {
        try{
            if (!liberouDados)
                wait();
        }catch(Exception e){
            e.printStackTrace();
        }
        return dadosWsdI;
    }

    public synchronized Vector getDadosConfiguracoes() {
        try{
            if (!liberouDados)
                wait();
        }catch(Exception e){
            e.printStackTrace();
        }
        return dadosConfiguracoes;
    }

    public synchronized void carregado() {
        try{
            liberouDados = true;
            notify();
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    private boolean ehWsdI(){
        return nomeArquivo.charAt(nomeArquivo.length()-1) == 'I';
    }

    public void setNomeArquivo(String nomeArquivo) {
        this.nomeArquivo = nomeArquivo;
    }
}

```

## Classe WsdIParser

```
package comunicacao;
```



```

import java.io.InputStream;
import java.io.InputStreamReader;

import org.kxml2.io.KXmlParser;
import org.xmlpull.v1.XmlPullParser;

public class WsdlParser {

    protected KXmlParser parser;
    protected InputStream entrada;

    private String namespace;
    private String nomeAtributo;

    public WsdlParser(String namespace, String nomeAtributo){
        this.parser = new KXmlParser();
        this.entrada = null;
        this.namespace = namespace;
        this.nomeAtributo = nomeAtributo;
    }

    public String getValorAtributo(){
        String valor = "";
        try{
            parser.setInput(new InputStreamReader(entrada));
            do{
                parser.nextTag();
            }while(!parser.getName().equals(namespace) );
            parser.require(XmlPullParser.START_TAG, null, namespace);
            valor = parser.getAttributeValue("",nomeAtributo);
            entrada.close();
            parser.setInput(null);
        }catch(Exception e){
            e.printStackTrace();
        }
        return valor;
    }

    public void setEntrada(InputStream entrada) {
        this.entrada = entrada;
    }

    public void setAlvos(String namespace, String nomeAtributo){
        this.namespace = namespace;
        this.nomeAtributo = nomeAtributo;
    }
}

```

# Artigo

## Desenvolvimento de uma Aplicação para Provimento de Web Services em Redes Ad Hoc

Fernando Yuji Miyakawa

Ciências da Computação, 2006/1

Departamento de Informática e Estatística

Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-900

[miyakawa@inf.ufsc.br](mailto:miyakawa@inf.ufsc.br)

### RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação para fornecer e executar web services em redes Ad Hoc. A idéia principal é permitir que usuários dessa aplicação possam ter acesso a serviços que geralmente estão disponíveis apenas na rede fixa, ou ainda, necessitam dela no momento da sua utilização.

### ABSTRACT

This work shows us the development of an application to supply and to execute web services in Ad Hoc networks. The main idea is to allow that the user of this application could have access to services that usually are available only in a fixed network, or even, when they need it in the moment of its uses.

## Introdução

Têm-se como objetivos gerais o desenvolvimento de uma aplicação que possa disponibilizar a arquitetura *web services* em redes *Ad Hoc* e de maneira *Ad Hoc de facto*, ou seja, o usuário acessa o serviço sem o auxílio da rede fixa, explorando assim, em sua totalidade, as características da rede *Ad hoc*.

Há um grande interesse da comunidade científica em disponibilizar serviços aos usuários das redes móveis. Esses usuários possuem uma grande carência de serviços, uma vez que tais serviços encontram-se geralmente na rede fixa, ou dependem dela no momento de sua execução.

## Computação Móvel

O termo computação móvel não é um conceito ainda bem definido, e envolve elementos como hardware, dados, aplicações e usuários que têm a capacidade de se moverem para diferentes localizações durante o curso da computação. Dentre alguns desafios podemos citar o difícil acesso à serviços em ambientes

móveis, desenvolvimento de protocolos de suporte a computação móvel, segurança, entre outros.

Destacam-se as seguintes tecnologias:

### Padrão IEEE 802.11

O padrão de comunicação IEEE 802.11 foi criado em 1999 para suportar a comunicação em redes locais sem fio, (WLANs – *Wireless Local Networks*). A especificação define uma camada de acesso ao meio, camada MAC, e diferentes camadas físicas, tornando possível acessar o meio de três formas possíveis: FHSS (*Frequency Hopping Spread Spectrum*), DSSS (*Direct Sequence Spread Spectrum*) e infravermelho.

### Bluetooth

A proposta da tecnologia *Bluetooth* é habilitar os usuários para a conexão de uma grande variedade de dispositivos de computação e telecomunicações de maneira simples e fácil, sem a necessidade de carregar, comprar ou conectar cabos. Está globalmente disponível e tem compatibilidade mundial, portanto, sendo um padrão global para conectividade *wireless*.

## WAP

É uma arquitetura de protocolos que permite que dispositivos móveis executem aplicações através de redes *wireless*, fazendo acesso aos serviços disponíveis no ambiente Internet. Exemplos destes dispositivos são os telefones celulares, PDAs e *paggers*.

## **Redes Ad Hoc**

Numa rede *ad hoc*, ou MANETs (*Mobile Ad hoc Network*), os nós ou nodos possuem a capacidade de se comunicarem independentemente de qualquer infra-estrutura física de comunicação ou administração centralizada, criando uma rede "*on the fly*", na qual alguns dos dispositivos da rede fazem parte da rede de fato apenas durante a duração da sessão de comunicação, ou, no caso de dispositivos móveis ou portáteis, enquanto estiverem a uma certa proximidade do restante da rede.

Assim sendo, essas redes são rapidamente estabelecidas e altamente flexíveis, podendo chegar

a lugares difíceis ou até mesmo inacessíveis às redes infra-estruturadas. São indicadas principalmente em situações onde não se pode, ou não faz sentido, instalar uma rede fixa (SADOK & D'OLIVEIRA, 2001). Algumas vantagens são: instalação rápida, tolerância à falhas, conectividade e mobilidade e já as desvantagens: roteamento, localização, taxa de erros, entre outras.

## **Web Services**

*Web services* é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato *XML*.

*Utilizam* tecnologias programáveis e reutilizáveis que aproveitam a flexibilidade da *Internet*. Com eles é possível ter uma infinidade de aplicativos

conectados em rede, mesmo rodando em plataformas diferentes, fornecendo informações a todos os seus clientes, parceiros e funcionários. Baseiam-se num conjunto de padrões abertos, incluindo SOAP, WSDL e UDDI.

### WSDL

WSDL (Web Services Description Language) é um meio de descrever as funcionalidades de um web service, informando as funções que cada serviço pode prestar (descrição da implementação), que informações de entradas são necessárias para que o serviço possa ser executado (descrição da interface), quais os tipos de resultados devem ser esperados (também na descrição da interface).

### SOAP

SOAP (Simple Object Access Protocol) é um protocolo baseado em XML para troca de informação em um ambiente distribuído e descentralizado. Produz um envelope que define um framework para descrever qual é a mensagem e como processá-la.

### UDDI

Especificação que foi desenvolvida para um registro central de acesso e controle, que descreve como os dados devem ser armazenados em um registro, além de definir os métodos possíveis para publicação e busca desses registros. Os dados incluem os contatos da empresa, como se comunicar com eles, uma lista de serviços disponíveis e como usá-los por meio de algum tipo de programação (OASIS, 2003).

## **Proposta do Projeto e Resultados**

### O Modelo Web Services em Redes Móveis Ad Hoc

Segundo os autores (FARIA et al., 2004), os dispositivos wireless se tornam servidores para seus pares, habilitando-os a fornecer conteúdo, roteamento do tráfego na rede, e muitos outros serviços.

#### *Conexão com a rede fixa*

O servidor conectado à rede fixa contém os módulos de software necessários para suprir um

dispositivo móvel que queira tornar-se fornecedor de serviço para seus pares em uma rede Ad hoc. Denominamos esses módulos de módulo móvel e módulo fixo (FARIA et al., 2004).

O módulo móvel do servidor conectado à rede fixa contém um repositório UDDI privado contendo todos os serviços que esse servidor pode disponibilizar, e mais a aplicação fornecedora e a aplicação cliente de cada serviço publicado nesse repositório UDDI.

O usuário do DMF (Dispositivo Móvel Fornecedor) dispara na rede no qual ele está localizado, uma mensagem broadcast oferecendo a seus pares o serviço anteriormente selecionado

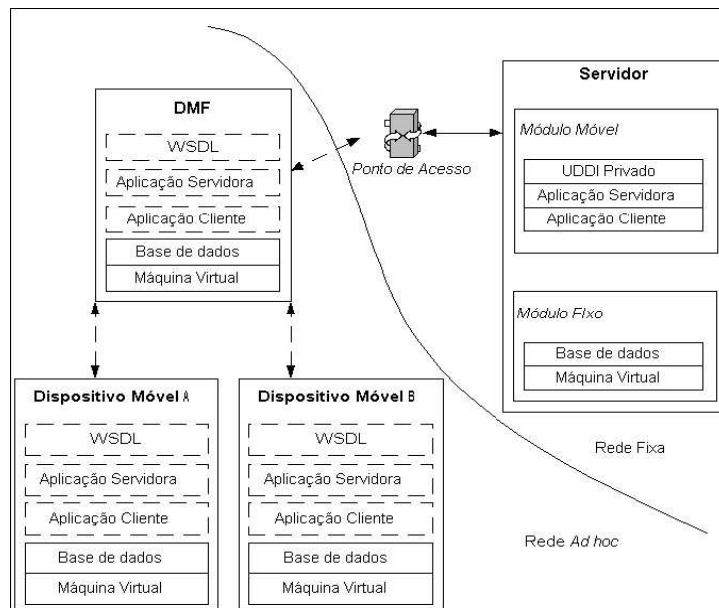
e adquirido através da rede fixa (FARIA et al., 2004).

Os dispositivos que compõem a rede recebem a mensagem enviada pelo DMF e optam entre aceitar ou recusar o serviço. Os dispositivos interessados retornam a mensagem ao DMF e tornam-se nesse momento dispositivos clientes, estabelecendo assim, uma conexão com o DMF requisitando o módulo móvel (FARIA et al., 2004).

Uma vez terminada a transferência dos arquivos necessários, a execução do serviço será realizada localmente, minimizando assim, alguns dos desafios inerentes ao ambiente móvel.

| Etapa | Ação   |
|-------|--|
| A     | O DMF dispara uma mensagem broadcast.  |
| B     | Os dispositivos A e B solicitam o serviço ao DMF.  |
| C     | Será estabelecida uma conexão entre DMF e dispositivos A e B para a transferência do módulo móvel. |
| D     | Execução local do serviço.   |

**Etapas do funcionamento do modelo**



**Modelo Utilizado**

## Funcionalidades da aplicação

### *Oferta de serviço*

Para se oferecer um serviço a outro nó, o usuário deverá escolher um serviço e confirmar, sendo assim, será enviada uma mensagem broadcast, que é nada mais o arquivo wsdl do respectivo serviço que, com isso, o usuário se informe sobre o serviço antes de baixá-lo, a todos os nós ativos da rede Ad Hoc. Em cada nó da rede será atualizada a lista de serviços disponíveis na rede assim que chegar a mensagem.

### *Download de serviço*

Através dessa funcionalidade, o usuário poderá baixar para seu dispositivo, serviços

disponíveis na rede Ad Hoc que pode ser visualizada na lista de serviços disponíveis na rede. Ao escolher um item da lista e confirmar, será enviado o nome do serviço requisitado ao DMF (Dispositivo Móvel Fornecedor), que através deste envio, terá conhecimento do serviço propriamente dito a ser enviado de volta ao nó que pediu o serviço. É possível baixar ao mesmo tempo vários serviços, tanto de servidores diferentes como do mesmo servidor.

### *Envio do serviço desejado*

O envio do serviço é automaticamente carregado e enviado pela aplicação, tendo em vista que na mensagem, do nó que

deseja um serviço, está contido o nome do serviço desejado.

É possível também fornecer um ou mais serviços ao mesmo tempo, para diferentes clientes ou para o mesmo.

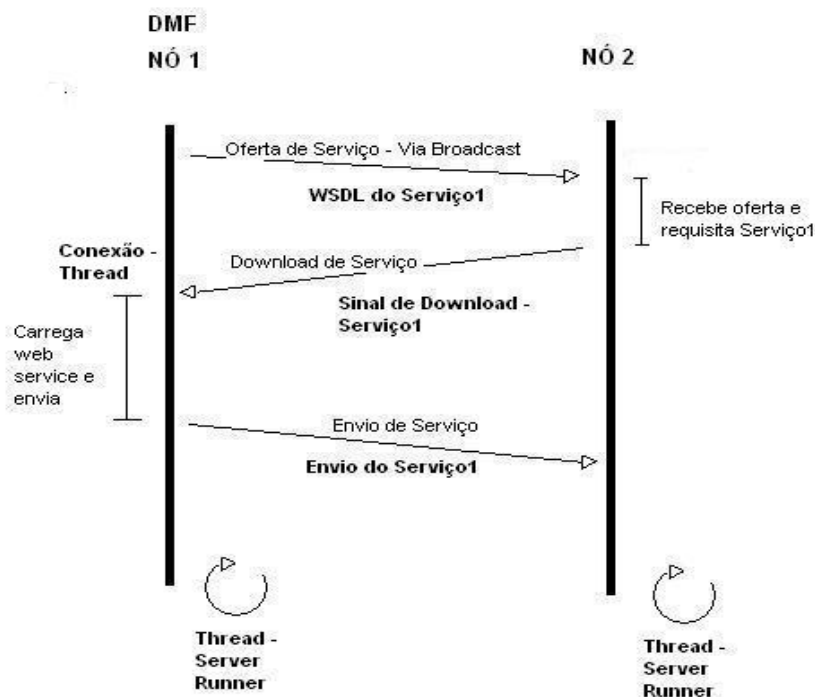
### Informações dos serviços

É possível obter informações de todos os serviços disponíveis na rede e localmente, já que logicamente os arquivos wsdl de cada serviço encontram-se localmente e foram oferecidos em algum momento anterior. No arquivo wsdl pode ser encontrado informações como, o endereço do

serviço no qual pode ser encontrado, protocolo de transporte utilizado, entre outras.

### ServerRunner

Entidade responsável pela realização e disponibilidade das conexões. Funciona da seguinte maneira: quando iniciado, permanece na espera até um cliente se conectar; cria uma thread de servidor socket e atribui a conexão que acabou de ser estabelecida à thread; retorna ao estado de espera por uma nova conexão de um cliente.



Visão geral das funcionalidades



## Testes e Resultados

Apesar da inviabilidade dos testes em dispositivos reais, os testes no simulador foram bem sucedidos e, assim como foi demonstrado, a utilização da aplicação é uma alternativa interessante para o fornecimento de web services a usuários de redes ad hoc. A aplicação torna factível levar serviços a lugares inacessíveis a rede fixa, lugares onde a necessidade é temporária, buscando de fato, uma computação disponível a qualquer hora e em qualquer lugar.

## Conclusões e Trabalhos

### Futuros

Embora tenhamos muitas limitações e uma série de melhorias e recursos adicionais a serem aderidos à aplicação, a adoção da aplicação para o fornecimento de serviços móveis minimiza os diversos desafios encontrados quando desejamos levar serviços aos usuários destes ambientes tornando-se um diferencial dentro da topologia Ad hoc por permitir a

execução dos serviços sem auxílio de qualquer infra-estrutura fixa.

Como trabalhos futuros, seria interessante a implementação de um mecanismo que possa medir a taxa de transmissão de downloads e uploads de dados e a largura de banda disponível nas conexões realizadas. Uma outra necessidade seria que a aplicação possa permitir o controle do número máximo de conexões ativas de modo automatizado para que o dispositivo não fique sobrecarregado. Além dessas questões, poderia existir suporte a alta disponibilidade do servidor de serviços da aplicação e segurança das conexões de troca de mensagens.

### Referências Bibliográficas

CHAPPELL, D.; JEWELL, T. **Java Web Services**. O'Reilly, 2002.

DANTAS, Mário. **Tecnologias de redes de comunicação e computadores**. Rio de Janeiro: Axcel Books, 2002.

FARIA, F.; MAZZOLA, V.; DANTAS M.A.R. **Servidores Móveis em Redes Ad hoc**. II Escola Regional

de Redes de Computadores, 129-134, Canoas - RS. Jul. 2004.

MATEUS, G. R.; LOUREIRO, A. A. F. **Introdução a Computação Móvel**. 1.ed. DCC/IM, COPPE/Sistemas, NCE/UFRJ, 11a. Escola de Computação, 1998.

MATEUS, G.R.; LOUREIRO, A. A. F. **Introdução à Computação Móvel**. 2.ed. Disponível em [http://www.dcc.ufmg.br/~loureiro/cm/docs/cm\\_livro\\_2e.pdf](http://www.dcc.ufmg.br/~loureiro/cm/docs/cm_livro_2e.pdf)>. Acesso em: 10/05/2006.

OASIS Member Section, **Universal Description, Discovery e Integration** <http://uddi.org/pubs/uddi-v3.0.1-20031014.htm>>. Disponível on-line, acesso em 14/11/2005.

SADOK, D. H. F.; D'OLIVEIRA, S. T. J. **Análise de Tráfego de Dados em Redes Bluetooth**. 2001. 51 f. Centro de Informática, Universidade Federal de Pernambuco, Recife, 2001.