UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ARQUITETURA DE CONECTIVIDADE PARA DISPOSITIVOS MÓVEIS NA PLATAFORMA JAMP

LUIZ ANTONIO FALAGUASTA BARBOSA

São Carlos – SP
Outubro de 2006

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

B238ac

Barbosa, Luiz Antonio Falaguasta.
Uma arquitetura de conectividade de dispositivos móveis na plataforma JAMP / Luiz Antonio Falaguasta Barbosa. -- São Carlos : UFSCar, 2007.
83 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2006.

1. Sistemas distribuídos. 2. Computação ubíqua. 3. Java/RMI. 4. Middleware. 5. Plataforma JAMP. I. Título.

CDD: 005.43 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Uma Arquitetura de Conectividade para Dispositivos Móveis na Plataforma JAMP"

LUIZ ANTONIO FALAGUASTA BARBOSA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Luis Carlos Trevelin
(Orientador – DC/UFSCar)

Profa. Dra. Regina Borges de Araujo
(DC/UFSCar)

Prof. Dr. Eleri Cardozo
(DCA/FEE/UNICAMP)

São Carlos
Dezembro/2006

AGRADECIMENTOS

Agradeço a Deus por ter oferecido todas as oportunidades que me levaram à efetivação desse trabalho.

Agradeço a meus pais Maria Angélica Falaguasta Barbosa e Antonio Barbosa, pelo milagre da vida e por todos os seus ensinamentos.

Agradeço à minha amada Patrícia, que teve paciência, compreensão e me deu força em todos os momentos.

Agradeço ao meu grande amigo Prof. Dr. Rodrigo Fernandes de Mello, que me colocou em contato com o maravilhoso mundo dos Sistemas Distribuídos em suas aulas, na graduação.

Agradeço ao meu orientador Prof. Dr. Luis Carlos Trevelin, que me motivou e mostrou-me o caminho para obter êxito nessa jornada.

Aos meus amigos Luciano Bernardes de Paula, o Ruivo, Roberto Rigolin de Souza Lopes e Leandro M. Maciel, que ajudaram, opinaram e disseram coisas engraçadas durante o almoço, nos bate-papos no meio da tarde em frente ao departamento, entre a leitura e escrita de um artigo e outro, entre a composição de uma linha código e outra.

RESUMO

Esta dissertação de mestrado trata dos aspectos envolvidos para a criação de um *framework* de conectividade na JAMP *(Java Architecture for Media Processing*). A concepção desse *framework* tem por objetivo possibilitar a utilização de dispositivos móveis por meio de outras tecnologias de rede sem fio. Tais tecnologias, como Bluetooth, usadas para a publicação/localização de serviços registrados no JBroker, *broker* da Plataforma JAMP desenvolvido em Java/RMI, bem como a comunicação de dispositivos que dispõem de tal tecnologia, foram empregadas na implementação e testes do *framework* desenvolvido.

A implementação visa ao acesso, a partir de dispositivos com interface de rede Bluetooth, a serviços remotos disponíveis na Internet. Esses serviços são requisitados por dispositivos móveis, tais como PDAs, celulares ou *smartphones*, que consultam pontos de acesso, via Bluetooth. Os pontos de acesso consultam o JBroker que retorna uma referência para o serviço a ser consumido. Os pontos de acesso então encaminham o resultado aos dispositivos móveis.

Os experimentos desenvolvidos neste trabalho basearam-se no cenário onde um dispositivo móvel, em comunicação Bluetooth, requisita serviços a um ponto de acesso em um PC e este, via TCP/IP, invoca métodos remotos nos servidores que disponibilizam os serviços, via Java/RMI.

A JAMP já apresentava todo o mecanismo de localização de serviços via Java/RMI, porém não apresentava a extensão desenvolvida neste trabalho. Nesta, foram criados um *proxy*, que é executado no ponto de acesso, e a utilização de serviços da JAMP a partir de dispositivos móveis que se comunicam através de tecnologia de rede diferente de TCP/IP. Sendo assim, sua inovação está na capacidade de utilização de dispositivos que dispõem de tecnologia de rede Bluetooth, podendo ser estendido a outras tecnologias, sem necessitar da reengenharia previamente existente na arquitetura.

Palavras-Chave: Computação Ubíqua, RMI, *Middleware*, Plataforma JAMP.

A Connectivity Archtecture for Mobile Devices in JAMP Platform

ABSTRACT

This master thesis deals with the involved aspects for the creation of a framework for connectivity in JAMP (Java Architecture for Media Processing). The conception of this framework aims to make possible the use of mobile devices by means of other wireless network technologies. Such technologies, as Bluetooth, used for the publication/localization of services registered in the JBroker, the broker of JAMP Platform developed in Java/RMI, as well as the communication of devices that make use of such technologies through this broker, had been used in the implementation and tests of the framework developed.

The implementation aims to access, from devices with Bluetooth network interface, the available remote services in the Internet. These services are requested by mobile devices, such as PDAs, cellular or smartphones, that they consult access points, in Bluetooth communication. The access points consult the JBroker that returns a reference for the service to be consumed. The access points then direct the result to the mobile devices.

The experiments developed in this work had been based on scenes where a mobile device, in Bluetooth communication, requests services to a point of access in a PC and this, saw TCP/IP, invokes remote methods in the servers who disponibilizam the services, saw Java/RMI.

JAMP already all presented the mechanism of localization of services saw Java/RMI, however it did not present the extension developed with this work, where they had been created one proxy, that it is executed in the access point, and the use of services of the JAMP to break mobile devices that if communicate through technology of different net of TCP/IP. Being thus, its innovation is in the capacity of use of devices that make use of technology of Bluetooth net, being able to be extended to other technologies, without needing would reengeneering previously existing in the architecture.

Keywords: Ubquitous Computing, RMI, Middleware, JAMP Platform.

LISTA DE FIGURAS

Figura 1: Dispositivo Bluetooth [SIG 01]. 13
Figura 2: Especificação da Pilha de Protocolo Bluetooth [SIG 01]. 14
Figura 3: Salto de freqüência por divisão de tempo [AUS 00]. 15
Figura 4: Duas piconets conectadas formando uma scatternet [TAN 03]. 16
Figura 5: Interação da API Java para Bluetooth com a pilha Bluetooth [SUN BLUETOOTH]. 20
Figura 6: (a) Smartphone Treo 650, (b) PDA LifeDrive e (c) celular L7. 23
Figura 7: J2ME frente às demais edições Java [FMJ 04]. 32
Figura 8: Aplicação na arquitetura J2ME. 33
Figura 9: Pilha de protocolos do Universal Plug and Play [HEL 02]. 37
Figura 11: Arquitetura JAMP. 50
Figura 12: Processo de Trading. (1) Publicação do serviço. (2) Consulta ao JBroker. (3) Referência do objeto de serviço. (4) Acesso direto ao serviço. 51
Figura 13: Arquitetura JAMP e suas camadas. 51
Figura 14: Cenário de operação do framework de conectividade BConn. 57
Figura 15: Funcionamento do Surrogate [SUR 06]. 58
Figura 16: Diagrama de casos de uso do framework de conectividade BConn. 60
Figura 17: Diagrama de classes do framework BConn. 60
Figura 19: Seqüência de eventos de funcionamento do BConn no acesso a serviços da JAMP. 63
Figura 20: Cenário utilizado no estudo de caso. 68

LISTA DE TABELAS

Tabela 1: Versões do 802.16 [WIMAX]. 13
Tabela 2: Protocolos e camadas da pilha de protocolos Bluetooth [HEL 01]. 14
Bluetooth Rádio, Baseband, LMP, L2CAP, SDP 14
Tabela 3: Classes de potência. 15
Tabela 4: Os perfis do Bluetooth, adaptado de [TAN 03]. 19
Define o relacionamento cliente/servidor para movimentação de objetos 19
Tabela 5: Medidas de tempo para descobrimento de dispositivo e de serviço. 71

LISTA DE ABREVIATURAS E SIGLAS

JAMP	Java Architecture for Media Processing
SO	Sistema Operacional
RMI	Remote Method Invocation
API	Application Program Interface
TCP/IP	Transmission Control Protocol / Internet Protocol
LAN	Local Area Network
WECA	Wireless Ethernet Compatibility Alliance
ISM	Industrial, Science, and Medical
FCC	Federal Communications Commision
DSSS	Direct-Sequence Spread Spectrum
FHSS	Frequency-Hopping Spread Spectrum
IrDA	Infrared Data Association
WMAN	Wireless Metropolitan Area Network
PAN	Personal Area Networks
FHSS	Frequency Hopping Spread Spectrum
TDD	Time Division Duplexing
GFSK	Gaussian Frequency Shift Keying
ACL	Asynchronous Connectionless
SCO	Synchronous Connection-Oriented
PCM	Pulse Code Modulation
CVSD	Continuous Variable Slope Delta
LMP	Link Manager Protocol
L2CAP	Logical Link Control and Adaptation Protocol
SDP	Service Discovery Protocol
RFCOMM	Radio Frequency Communications Protocol
MTU	Maximum Transmition Unit
UMTS	Universal Mobile Telecommunications System

GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
CDMA	Code Division Multiple Access
WCDMA	Wideband Code Division Multiple Access
LEP	Light-emitting Polimer
CoG	Chip over Glass
LCoG	Liquid Crystal over Glass
COM	Cryptographic Provider Manager
PAP	Password Authentication Protocol
CHAP	Challenge-Handshake Authentication Protocol
PPP	Point-to-point Protocol
UDP	User Datagram Protocol
OBEX	Object Exchange
WAP	Wireless Application Protocol
vCard	Virtual Card
vCal	Virtual Calendar
IrMC	InfraRed Mobile Communication
WAE	Wireless Application Environment
TDD	Time Division Duplexing
GFSK	Gaussian Frequency Shift Keying
PCM	Pulse Code Modulation
CVSD	Continuous Variable Slope Delta
LMP	Link Manager Protocol
L2CAP	Logical Link Control and Adaptation Protocol
MVJ	Máquina Virtual Java
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
J2ME	Java 2 Micro Edition
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
MIDP	Mobile Information Device Profile
XML	eXtensible Markup Language
W3C	World Wide Web Consortium
SGML	Standard Generalized Markup Language
SLP	Service Location Protocol

SDK	Software Development Kit
OSGi	Open Services Gateway initiative
SSDP	Simple Service Discovery Protocol
SOAP	Simple Object Access Protocol
URL	Universal Resource Locator
IETF	Internet Engineering Task Force
UDDI	Universal Descripton, Discovery, and Integration
WSDL	Web Services Description Language
SQL	Structured Query Language
TP	Transaction Process
RPC	Remote Procedure Call
MOM	Message Oriented Monitor
ORB	Object Request Broker
SOMA	Secure and Open Mobile Agent
QoS	Quality of Service
SSL	Secure Socket Layer
UML	Unified Modeling Language
PC	Personal Computer
KVM	Kilobyte Virtual Machine
MIDlet	Mobile Information Device Profile let
CDC	Connected Device Configuration
SGBD	Sistema Gerenciador de Banco de Dados
MVC	Model View Controller

SUMÁRIO

1 INTRODUÇÃO 1
2 COMPUTAÇÃO UBÍQUA 4
2.1 Conceitos 4
2.2 Características da Computação Ubíqua 6
2.3 Aplicações 7
2.4 Tecnologias 8
2.4.1 Tecnologias de rede 8
2.4.1.1 Mobilidade Ad Hoc 10
2.4.1.2 Bluetooth 13
2.4.1.3 Rede celular 20
2.5 Dispositivos 22
2.5.1 Tipos de dispositivos 23
2.5.2 Componentes dos dispositivos 24
2.5.2.1 Processadores 24
2.5.2.2 Memória 24
2.5.2.3 Telas 25
2.5.2.4 Bateria 25
2.6 Software 26
2.6.1 Sistemas operacionais 26
2.6.1.1 Palm OS 27
2.6.1.2 Linux embarcado 28
2.6.1.3 Windows CE 29
2.6.1.4 TinyOS 29
2.6.2 Linguagens de programação para dispositivos móveis. 31
2.6.2.1 Java 31
2.6.2.2 XML 33
2.6.3 Técnicas para a obtenção da Ubiqüidade 34

2.6.3.1 Ciência de Contexto 34
2.6.3.2 Adaptação de Conteúdo 34
2.7 Descobrimento de recursos 35
2.7.1 Jini 35
2.7.1.1 Tecnologias no mercado 36
2.7.2 UPnP 37
2.7.3 Salutation 38
2.7.3.1 Serviço de registro 38
2.7.3.2 Serviço de descobrimento 38
2.7.3.3 Serviço de disponibilidade 39
2.7.3.4 Serviço de gerenciamento de sessão 39
2.7.3.5 Salutation-Lite 39
2.7.3.6 Service Location Protocol 39
2.7.4 UDDI 39
2.7.5 Bluetooth SDP 40
2.8 Desafios 41
2.9 Conclusões 42
3 MIDDLEWARE 44
3.1 Tipos de middleware 48
3.2 JAMP - Java Architecture for Media Processing 49
3.3.1 Trabalhos relacionados 53
4 O FRAMEWORK BCONN 55
4.1 Cenário de operação do framework 56
4.2 Concepção do BConn 57
4.3 Funcionamento do BConn 62
5 ESTUDO DE CASO 66
5.1 Descrição do cenário de teste 66
5.2 Descrição do sistema 68
5.3 Análise dos resultados do estudo de caso 69
5.3.1 Facilidade de uso e funcionalidade do framework 69
5.3.2 Desempenho 70
6 CONSIDERAÇÕES FINAIS 73
6.1 Conclusões 73
6.2 Trabalhos Futuros 74
7 REFERÊNCIAS 76

1 INTRODUÇÃO

Computação ubíqua é voltada à interação de usuários, dispositivos móveis e fixos, em ambientes providos de poder computacional, capturando informações, processando-as e disponibilizando-as aos usuários, de modo a tornar o ambiente personalizado àqueles que o habitam. Trata da permeabilidade de dispositivos computacionais no ambiente, da mobilidade de usuários e dispositivos e da criação de modelos computacionais que suportem tal dinâmica.

Para que haja tal interação, recursos computacionais e de comunicação são embarcados em equipamentos que fazem parte dos diferentes ambientes do usuário – de forma a potencializar o cenário – sem ser foco de atenção do mesmo, como afirma Weiser em [WEI 91]. Com isso, poder computacional e de comunicação está presente em utensílios domésticos, em equipamentos para controle e monitoração de linhas de produção de fábricas e em ambientes externos, constituindo um ambiente onde há presença da Computação Ubíqua [WEI 91].

Computação Ubíqua trata da criação de modelos computacionais dinâmicos constituídos a partir da mobilidade do usuário em novos ambientes ou da utilização de modelos na revisitação dos ambientes [LYY 02a]. Para tanto, arquiteturas que envolvem conceitos como *middleware* e objetos distribuídos podem ser explorados para a constituição de tais modelos, bem como a comunicação através de *sockets* [STE 97]. Ou seja, *middleware* é usado como plataforma que trata das funções necessárias para a constituição de modelos dinâmicos – tais como comunicação de rede, confiabilidade, sincronização e escalabilidade, oferecendo tais funções para que os modelos dinâmicos operem –, objetos distribuídos são usados para que equipamentos presentes no ambiente do usuário acessem serviços remotos e *sockets* para que uma estrutura bem definida de comunicação em rede seja empregada.

Tecnologias providas pela linguagem de programação Java, como API RMI [SUN RMI], que permite a invocação remota de métodos – uma evolução da RPC – presentes em outros dispositivos, e arquiteturas para a integração com outras linguagens, como CORBA

[CORBA], que permite a comunicação de objetos distribuídos implementados em diferentes linguagens de programação, constituem o entrelaçamento de ambientes heterogêneos para a formação de comunidades *ad hoc* de dispositivos. Essas comunidades, onde dispositivos móveis formam, de maneira momentânea, uma rede que utiliza os nós presentes entre pares que estabelecem comunicação, são utilizadas nos modelos dinâmicos para operar no ambiente. Além de RMI e CORBA, outras formas de comunicação entre dispositivos também devem ser empregadas para garantir a utilização de dispositivos com maiores restrições de processamento e taxa de transmissão de dados em rede, tais como o modelo cliente-servidor em tecnologias de rede que não se baseiam no modelo TCP/IP, como Bluetooth.

Visando a entrelaçar diferentes plataformas de *hardware* e sistema operacional (SO), arquiteturas para computação ubíqua têm sido implementadas na forma de *middleware* [GAR 02, ROM 02] servindo de suporte à interação de dispositivos e usuários em ambientes pervasivos, e fazendo a ligação entre aplicativos e dispositivos permitindo que ambientes heterogêneos possam comunicar-se através de diferentes tecnologias de rede e inclusive possibilitando o chaveamento entre elas.

Com o objetivo de promover esse entrelaçamento de dispositivos, é apresentada uma extensão para a plataforma JAMP (*Java Architecture for Media Processing*), uma arquitetura de *middleware* baseada em *frameworks*, que permite o acesso de dispositivos móveis a serviços dessa plataforma. A JAMP é uma plataforma de sistemas distribuídos baseada em serviços remotos. Esses serviços são oferecidos por meio de um *broker*, o JBroker, onde são publicados os serviços. Uma vez que os serviços são publicados, aplicações clientes que necessitem deles requisitam-nos ao *broker* para que este devolva uma referência do serviço, que será consumido pelos clientes. A partir da referência, o cliente usa o serviço por meio de invocação remota dos métodos presentes no serviço (operações que o serviço desempenha), utilizando RMI (*Remote Method Invocation*), uma API (*Application Program Interface*) da linguagem Java, sobre conexões TCP/IP (*Transmission Control Protocol / Internet Protocol*).

O trabalho desenvolvido nessa dissertação é baseado na linguagem de programação Java por ser multiplataforma, tanto de *hardware* quanto de SO, e APIs dessa linguagem como Bluetooth e RMI. Tal trabalho permite a utilização do RMI a partir de uma conexão Bluetooth. Essa conexão Bluetooth acessa um ponto de acesso, um *proxy* Bluetooth-TCP/IP, que encaminha as requisições de serviços ao JBroker, seguido da mesma lógica de utilização do serviço em RMI. Dessa forma, a extensão proposta com o *framework* permite o entrelaçamento de diferentes dispositivos, viabilizando o acesso de dispositivos móveis a

serviços de *middleware* implementados em *framework* de conectividade na plataforma JAMP.

Os resultados mostraram que o *framework* criado proporciona a utilização da JAMP como plataforma de sistema distribuído para a interconexão de redes. O estudo de caso gerado, onde a tecnologia de rede Bluetooth foi empregada, permitiu a utilização de uma infra-estrutura de objetos distribuídos, implementada com Java/RMI, onde celulares podem acessar serviços publicados em um *broker* da JAMP. A interconexão é assegurada por um *proxy* que encaminha requisições Bluetooth para uma rede TCP/IP, onde a parte RMI do sistema é executada.

O restante desse trabalho está organizado da seguinte forma: o segundo capítulo trata da Computação Ubíqua, ou seja, os conceitos, aspectos e tecnologias que permitem a interação do usuário com o ambiente por meio de elementos computacionais que permeiam os cenários que esse usuário visita. O terceiro capítulo descreve os tipos de *middleware* existentes, classificando a JAMP e discutindo a interconexão de redes por meio de *middlewares*. O quarto capítulo apresenta o BConn, quais as ferramentas utilizadas no seu desenvolvimento e qual modelo foi usado como base na sua construção. O quinto capítulo aborda o estudo de caso realizado para validação do *framework,* qual o cenário utilizado nesse estudo, a descrição dos equipamentos, a disposição dos mesmos, os resultados obtidos e suas medições. O sexto capítulo trata das considerações finais, ou seja, as conclusões que o estudo proporcionou e as propostas para trabalhos futuros, na busca de aumentar as potencialidades do *framework*.

2 COMPUTAÇÃO UBÍQUA

Esse capítulo aborda os conceitos e tecnologias utilizados em Computação Ubíqua, as tecnologias de rede que compõem seus cenários, aplicações, tipos de dispositivos – sua composição e características para utilização em Computação Ubíqua, sistemas operacionais e linguagens de programação. É abordado também o descobrimento de serviço, técnica usada para localizar serviços em ambientes de sistemas distribuídos.

Segundo Weiser em [WEI 91], precursor da computação ubíqua, as mais profundas tecnologias são aquelas que desaparecem. Isso é o que ocorre quando a computação passa à sua terceira geração, deixando de ser centrada nos computadores pessoais, onde cada estação de trabalho é utilizada por uma única pessoa, e passa a permear o ambiente de modo a existirem vários dispositivos computadorizados, onde cada um deles pode interagir com mais de um usuário e os demais dispositivos [WEI 91].

Nesse capítulo são abordados conceitos e características, expostos exemplos de aplicações, apresentadas tecnologias, dispositivos e *software* empregados em Computação Ubíqua, bem como os desafios existentes na mesma. Toda a discussão sobre a Computação Ubíqua serve de introdução para apresentar o *framework* desenvolvido neste trabalho.

O estudo para a construção do *framework* teve suas bases na Computação Ubíqua, onde foi planejada uma maneira para incorporar novos dispositivos a JAMP e fazer com que eles interajam com o ambiente de forma pervasiva, ou seja, utilizando serviços que permeiam o ambiente, e também ubíqua, pois tais serviços são onipresentes por usarem a Internet como estrutura de distribuição.

2.1 Conceitos

A Computação Ubíqua utiliza fatores de mobilidade e permeabilidade ao meio físico, diretamente relacionados à Computação Móvel e à Pervasiva [LYY 02a].

Segundo Lyytinen em [LYY 02a], Computação Móvel diz respeito à capacidade de mover um serviço computacional junto do usuário que o utiliza, tornando o computador um dispositivo disponível a vários usuários, expandindo suas capacidades para diferentes tarefas,

independente da localização do dispositivo. Isso pode ser obtido através da miniaturização de dispositivos e do provimento de rede sem fio de banda larga. Embora haja fatores que viabilizem a computação móvel, há outros que limitam sua utilização, tais como a capacidade limitada de bateria, tamanho de tela e taxa de transmissão de rede.

A Computação Móvel pode ser desdobrada em três tipos de mobilidade: de usuário, de terminal e de serviço. A mobilidade de usuário diz respeito à capacidade do usuário se mover, utilizando um serviço computacional em algum dispositivo computacional. Mobilidade de terminal trata-se da possibilidade de mover o dispositivo computacional utilizado pelo usuário. Já a mobilidade de serviço refere-se a um serviço que migra entre diferentes dispositivos para atender às necessidades do usuário.

Lyytinen também explica Computação Pervasiva em [LYY 02a]. Ele diz que Computação Pervasiva significa que o computador apresenta a capacidade de obter informação do ambiente em que está inserido e usá-la para a construção de modelos computacionais dinâmicos. Isso ocorre nos ambientes onde o computador se encontra sem ser percebido pelos usuários. Nesse sentido, a infra-estrutura da JAMP pode ser usada para a criação de serviços de computação ubíqua, utilizando-se do serviço de conectividade – disponibilizado pelo *framework* desenvolvido neste trabalho – para viabilizar a interação de dispositivos móveis com o ambiente, ainda que eles não sejam imperceptíveis. Da mesma forma que os dispositivos obtêm informação do ambiente, eles também podem transformar o ambiente quando acoplados a outros equipamentos, resultando numa interação inteligente. Essa é a idéia básica da Computação Pervasiva, uma área povoada por sensores, atuadores, computadores móveis e fixos e modelos virtuais e físicos de ambientes físicos e cognitivos/sociais.

O principal desafio da Computação Pervasiva é o grande esforço envolvido para ensinar e/ou configurar os dispositivos do sistema a respeito do ambiente onde se encontram.

Computação Ubíqua significa que qualquer dispositivo computacional é capaz de construir modelos computacionais dinâmicos a partir de vários ambientes e configurar seus serviços de acordo com esses ambientes. Além disso, os dispositivos são capazes de se ajustarem a ambientes nos quais já operaram, auxiliando no trabalho de seus usuários quando estes retornam aos ambientes, ou acessando serviços ao entrar em novos ambientes.

Essa onipresença das aplicações de computação ubíqua é tipicamente obtida através do transporte dos dispositivos junto do usuário ou das aplicações movendo-se entre dispositivos, seguindo o usuário [BAN 02].

A transição para a computação ubíqua foca novos desafios técnicos, sociais e organizacionais. No nível técnico, há vários problemas ainda não resolvidos referentes a projeto e implementação de arquiteturas que permitem autoconfiguração de serviços de computação ubíqua em larga escala. Outros desafios também surgirão em termos de como se deve projetar e implementar um serviço ubíquo.

Computação Ubíqua é pautada no conceito de que só há ubiqüidade havendo capacidade de deslocamento, utilizando-se de redes de comunicação sem fio quaisquer e mantendo conexão de rede de dispositivos móveis que acompanham o usuário. Isso permite que o usuário continue a utilizar suas aplicações, podendo ainda adaptá-las em função dos dispositivos que dispõe à sua volta, baseado em informações de contexto do ambiente, ou seja, preferências do usuário e configuração de dispositivos presentes nesse ambiente.

Partindo deste princípio, fatores como convergência de redes com e sem fio, capacidade dos dispositivos móveis – poder de processamento, capacidade de bateria e tamanho de tela – e ciência de contexto, através da capacidade que a infra-estrutura de computação ubíqua possui para capturar o contexto do ambiente – redes de sensores sem fio e RFid –, constituem desafios para que a Computação Ubíqua torne-se factível.

2.2 Características da Computação Ubíqua

Computação Ubíqua apresenta características como diversidade dos dispositivos, descentralização dos mesmos – onde cada um deles tem uma utilização apropriada – e a conectividade, que deve estar presente independente da tecnologia de rede empregada na comunicação e da área de cobertura de cada uma delas [HAN 01]. A seguir, cada uma dessas características será abordada.

2.2.1 Diversidade

Os dispositivos móveis são voltados a aplicabilidades diferenciadas, portanto, diferente do PC que é voltado a atender *n* aplicações ao usuário, cada dispositivo móvel tem seu papel e, em alguns casos, os papéis de diferentes dispositivos se sobrepõem. Com isso, torna-se difícil oferecer aplicações comuns em dispositivos diferenciados já que eles têm suas próprias características e limitações.

2.2.2 Descentralização

Cada dispositivo tem seu papel, trabalhando de forma distribuída e assim construindo um ambiente de relações dinâmicas, onde cada um deles coopera um com o outro e quando os

dispositivos móveis necessitam atualizar informações com servidores, a sincronização torna-se relevante.

2.2.3 Conectividade

As aplicações e dispositivos acompanham os usuários de forma transparente entre redes heterogêneas. Para que isso se realize, é necessário que haja interoperabilidade entre redes de padrão aberto, garantindo a conectividade com a aplicação em qualquer lugar que o usuário esteja.

2.3 Aplicações

Aplicações para Computação Ubíqua tratam da interação com o usuário sem que este necessariamente faça uso da digitação de comandos em teclados, com clique de *mouse*, mas também através de comandos de voz e mesmo automaticamente, quando o sistema acompanha o usuário ou prevê ações do mesmo.

Abaixo são citados alguns exemplos de aplicações para Computação Ubíqua, embora haja muitos outros.

2.3.1 Casa inteligente

Esse tipo de aplicação é voltado à leitura de características dos moradores de uma casa e conseqüente autoconfiguração para que o ambiente adeque-se ao comportamento dos mesmos. São exemplos desse tipo de aplicação o Easy Living [EASY], projeto da Microsoft que trata do desenvolvimento de arquiteturas e tecnologias de ambientes inteligentes com foco em uma sala de estar; possui eletrodomésticos e serviços para controle de luminosidade, som e transferência de conteúdo de tela baseado na localização dos indivíduos. Adaptive House [MOZ 98] é um projeto da Universidade do Colorado que enfoca o desenvolvimento de um sistema, o ACHE (*Adapative Control of Home Environments*), para uma casa capaz de aprender com as ações diárias de seus habitantes, autoconfigurando-se e antecipando as necessidades dos usuários. Esse sistema foi desenvolvido para controlar o sistema de aquecimento, ventilação e ar condicionado, iluminação e água [ARA 03].

2.3.2 Ambiente de aprendizagem

Foi desenvolvido com o objetivo de prover suporte na produção de material didático para professores e auxílio à anotação de aulas de forma personalizada aos alunos. Alguns exemplos desse tipo de aplicação podem ser citados, tais como Classroom2000 [ABO 99], Lecture Browser [MUK 99] e Authoring on the fly [AUTHORING]. O Classroom2000, do Georgia Institute of Technology, compreende um conjunto de tecnologias de *hardware* e

software para a captura de aulas presenciais e posterior disponibilidade do material capturado sob a forma de hiperdocumentos multimídia customizados. Lecture Browser*,* um projeto da Universidade de Cornell*,* utiliza técnicas de visão computacional e combinação de mais de uma fonte de vídeo para produzir hiperdocumentos multimídia resultantes do processo de captura da aula ministrada pelo professor. Authoring on the Fly, projeto de suporte ao ensino da Universidade de Freiburg na Alemanha, difere de outros sistemas dessa natureza em função do poderoso modelo de sincronização das mídias capturadas da aula, das diversas formas de representação do material capturado, e das funcionalidades fornecidas para acesso posterior a esse material [ARA 03].

8

2.4 Tecnologias

As tecnologias relacionadas à mobilidade de usuários que se intercomunicam através de redes cabeadas, servindo de *backbone,* e de diversas tecnologias de rede sem fio, infra-estruturadas ou *ad hoc*, e de sensores, configuram o ambiente através de aplicações cientes de contexto e interoperam dispositivos que compõem os ambientes, sejam eles móveis ou fixos, são as que se destacam em Computação Ubíqua.

Para que haja mobilidade, além das questões relacionadas à comunicação de rede, existem ainda questões relacionadas à capacidade dos dispositivos móveis no que tange a autonomia de energia (limitação de baterias), tamanho de tela e poder de processamento, além de adaptação do conteúdo exibido [WAN 95].

2.4.1 Tecnologias de rede

Como citado acima, redes de comunicação podem ser classificadas em cabeadas ou sem fio e estas podem ainda ter seu mecanismo de rotas implementado em modo infra-estruturado ou *ad hoc*.

As redes sem fio infra-estruturadas são aquelas onde há uma hierarquia de dispositivos de rede, como roteadores, *switches* e *hubs*, que funcionam como dispositivos de distribuição do meio físico às estações de trabalho. Ou seja, existe um caminho definido por onde os dados trafegam, podendo-se assim elaborar mecanismos de segurança mais eficazes, em função do conhecimento do caminho que os pacotes tomam na rede. Estas redes são implementadas tanto na modalidade cabeada quanto na sem fio, onde a última é aquela cujo meio de acesso é estabelecido pelas ondas eletromagnéticas, permitindo então que qualquer indivíduo alheio à estrutura de comunicação tenha acesso aos pacotes por estes trafegarem num meio comum a todos; a atmosfera.

As redes *ad hoc*, por sua vez, utilizam os pontos finais de uma rede – estações de trabalho – como nós participantes da rota que pacotes tomam para trafegar de um ponto a outro da rede. As máquinas comunicam-se ponto-a-ponto e assim encaminham os pacotes, de modo a compor múltiplos saltos entre os nós da rede, no estabelecimento de uma rota. Tal modelo de comunicação é encontrado em redes sem fio, devido ao estabelecimento das rotas levarem em consideração a localização das estações à sua volta e, por estas poderem movimentar-se alterando o estado de localização e mudando as condições para o estabelecimento de rotas. As rotas tornam-se dinâmicas, apresentando caminhos diferentes para o estabelecimento de comunicação entre um dado par de nós em momentos diferentes, devido à mudança de topologia dos nós presentes entre a origem e o destino, em função de sua mobilidade.

Com a mobilidade requerida na computação ubíqua, a comunicação em redes sem fio é mais utilizada.

São requisitos contemplados na computação ubíqua, a mobilidade e conectividade para implementação da ubiqüidade dos sistemas. Baseados nisto, tais requisitos podem ser viabilizados através da ocorrência, nos dispositivos móveis, de diferentes tecnologias de rede que possibilitam aos mesmos permanecerem conectados mesmo que estes sejam levados a regiões onde não há cobertura de sinal de uma tecnologia, fazendo com que o dispositivo venha a utilizar outra tecnologia para manter-se conectado à aplicação que está executando no momento.

Dixit e Prasad afirmavam já em 2002 que no futuro haverá a era da mobilidade sem fronteiras *(seamless mobility)* através das redes sem fio heterogêneas [DIX 02, PRA 02]. A convergência de redes tem por objetivo possibilitar que o usuário mantenha a conexão de suas aplicações, podendo deslocar-se de um lugar para outro sem se preocupar com a autenticação em outra infra-estrutura de rede, tendo apenas que voltar sua atenção para a atividade que está desenvolvendo na aplicação que utiliza. Dentro dessa idéia, encontra-se o termo *seamless network* e *seamless mobility*, onde o primeiro diz respeito a uma rede que não exibe ao usuário as fronteiras que delimitam uma ou outra tecnologia. O usuário apenas utiliza a estrutura de que dispõe sem saber o que há por trás dela. E com a *seamless network*, tem-se também o conceito da *seamless mobility*, que significa a mobilidade que o usuário pode ter utilizando-se da *seamless network*. Ou seja, movendo-se através das redes, o usuário permeia sua conexão entre as diversas tecnologias de rede disponíveis no local sem ter que se preocupar com o estabelecimento e encerramento de conexão em uma ou outra tecnologia de rede.

Segundo Dixit em [DIX 03], a mobilidade é suportada somente na camada de acesso ao meio nas WLANs e redes celulares 2G e 3G, impedindo o *roaming* através do acesso entre redes heterogêneas e o roteamento entre domínios. Ao contrário disso, o suporte à mobilidade na camada de rede permitirá uma mobilidade global, onde a camada de acesso ao meio e a camada física serão completamente transparentes. Qualquer assunto relacionado à complexidade, custo e desempenho tornar-se-á irrelevante com os avanços nas tecnologias envolvidas. Há propostas de modelos de mobilidade em estudo no IETF como o IP Móvel (MIP) [PER 02], o *Cellular IP* (CIP) [CAM 99] e o *Network-based Localized Mobility Management* (NETLMM) [KEM, 06], cada um deles com seus prós e contras. Suportá-los requererá ao terminal móvel estar ciente de mobilidade e à pilha do protocolo IP do legado que tem sido implementada nele terá que ser substituída com as novas implementações de referência.

Com tal mobilidade, pode-se dizer que as redes que a implementam, redes sem fio *ad hoc*, são dotadas de ubiqüidade e dinamismo, sendo chamadas de redes *ad hoc* ubíquas. A computação ou rede ubíqua tem recebido grande atenção recentemente [OXYGEN]. Redes ubíquas referem-se à formação dinâmica e *ad hoc* de entidades colaborativas – pessoas e dispositivos –, que se adaptam a condições e tipos de redes, sendo alheias ao tipo de acesso. Configuram-se automaticamente quando nós e serviços aparecem, negociam, migram e desaparecem [DIX 02, OXYGEN, SNO 00, KAL 01].

2.4.1.1 Mobilidade Ad Hoc

Em redes *ad hoc* é muito importante que se tenha um protocolo de roteamento capaz de ajustar-se e adaptar-se a mudanças de topologia sem impor grande demanda nos recursos de rede disponíveis. Conexões sem fio, em comparação às com fio, possuem limitações de taxa de transmissão e os dispositivos móveis geralmente possuem capacidade limitada, como autonomia de bateria, disponibilidade de memória e poder computacional. Tais fatores adicionam pesadas demandas no protocolo de roteamento. No grupo MANET [MOBILE], há vários protocolos desenvolvidos ou em desenvolvimento, porém ainda não foi encontrado um protocolo que seja capaz de manipular todas as situações. Alguns deles funcionam bem sob determinadas situações e em outras, mal.

Diferentes tecnologias de comunicação sem fio podem ser utilizadas. Atualmente destacam-se a tecnologia Wi-fi ou IEEE 802.11, Bluetooth, IrDA, Redes de Sensores Sem Fio (RSSF) e rede celular (UMTS, GPRS, GSM, CDMA, WCDMA).

IEEE 802.11

O padrão IEEE 802.11 define protocolos que comandam todo o tráfego *wireless* baseado em *Ethernet*. Entretanto, dentro deste padrão, existem outras subdivisões que competem por um espaço no mercado [PEI 02].

Muito antes das redes sem fio despontarem no uso da *Ethernet*, o IEEE criou um grupo responsável pela certificação e padronização de LANs (*Local Area Network*). Este grupo foi intitulado de grupo 802, o qual trata da revisão de antigas e novas tecnologias de rede, para assegurar que estas são confiáveis e livres de conflito.

A certificação do 802 inclui muitos subconjuntos, os quais representam vários tipos de redes. Por exemplo, 802.3 é o padrão que define como a Ethernet funciona. Se um produto é Ethernet, ele deve atender todas as especificações deste padrão. O mesmo ocorre com o padrão 802.11.

Em conjunto com a caracterização dada anteriormente ao 802.11, existem certificações específicas como o 802.11a, 802.11b e 802.11g. Cada uma delas define diferentes métodos para prover *Ethernet* sem fio. Cada protocolo especifica vários aspectos de transferência de dados que os diferencia para outras certificações.

Extensões do 802.11

O 802.11 é uma série de padrões que define métodos de transmissão em tráfego de *Ethernet* sem fio. Comumente referenciada por tráfego Wi-Fi ou WLAN, esta tecnologia é testada e comercializada pelo WECA (*Wireless Ethernet Compatibility Alliance* – Aliança de Compatibilidade de *Ethernet* Sem Fio).

Este padrão possui vários sub-padrões importantes para o entendimento de WLANs – 802.11a, 802.11b e 802.11g. Cada um deles é baseado em diferentes camadas físicas e tem seus benefícios e desvantagens.

Na priorização da ratificação do padrão 802.11 pelo IEEE, várias outras tecnologias que usam diferentes formas de salto de espectro (*hopping spectrum*) foram desenvolvidas para facilitar a transferência de dados sem fio. Estas tecnologias eram proprietárias e tipicamente mais lentas do que os padrões finalizados, com velocidade entre 1 e 2 Mbps e freqüências de 900 MHz, 2.4 GHz.

Todos os padrões 802.11 utilizam freqüências ISM (*Industrial, Science, and Medical* – Industrial, Científico e Médico), como define o FCC (*Federal Communications Commision* – Comissão Federal de Comunicações) dos EUA, equivalente à Anatel no Brasil. Estas

freqüências, 900 MHz, 2.4 GHz e 5GHz, são faixas abertas que podem ser usadas por qualquer pessoa para testes ou comercialização sob autorização do órgão responsável.

Após a ratificação do IEEE em 1997, três principais tecnologias de freqüência tornaram-se os principais métodos de transferência de dados: DSSS (*Direct-Sequence Spread Spectrum*)**,** FHSS (*Frequency-Hopping Spread Spectrum*) e IrDA (*Infrared Data Association*). Destas três, o DSSS e o FHSS mostraram-se mais eficazes e foram incorporados a muitas tecnologias WLAN.

IEEE 802.11b

O padrão 802.11b trata do envio de sinais *wireless* usando seqüência direta de espalhamento de espectro (DSSS) numa faixa de 2.4 GHz.

A atual implementação do padrão 802.11b suporta velocidades de 1 a 2 Mbps, encontrada no padrão WLAN antigo, 802.11, usando DSSS numa faixa de 2.4 GHz. O padrão atual adiciona faixas de 5.5. e 11 Mbps para operação.

IEEE 802.11g

O padrão 802.11g apresenta como principal diferença a velocidade de 54 Mbps, sendo que as demais características mantêm-se usando seqüência direta de espalhamento de espectro (DSSS) em freqüência de 2.4 GHz.

IEEE 802.16

Voltado à utilização em redes metropolitanas sem fio (WMAN), o padrão IEEE 802.16, também conhecido como WiMax, é uma rede capaz de atender até 1000 usuários por estação base, operando a uma taxa de transferência máxima de 75 Mbps e com alcance de até 48 km [EKL 02].

O padrão definido em 2001, voltava-se à utilização na faixa de freqüência de 10 a 66 GHz e taxa de transferência de 34 Mbps com linha de visada. Em 2003, com a versão 802.16a, não é mais necessário a linha de visada e a faixa de freqüência fica entre 2 e 11 GHz. Outros detalhes estão presentes na Tabela 1.

Tabela 1: Versões do 802.16 [WIMAX].

IEEE 802.16 Dezembro de 2001	IEEE 802.16c Dezembro de 2002	IEEE 802.16a Janeiro de 2003	IEEE 802.16d 1º Trimestre de 2004 WIMAX	IEEE 802.16e 4º Trimestre de 2004
10-66 GHz Linha de visada Até 34 Mbps (canalização de 28 MHz)	Interoperabilidade	2-11 GHz Sem linha de visada Até 75Mbps (canalização de 20 MHz)	Modificações na 802.16a e interoperabilidade	WIMAX Mobilidade Nômade 802.11/16

2.4.1.2 Bluetooth

Bluetooth é uma tecnologia de comunicação de rede de curta distância voltada para aplicações de redes pessoais (PAN – *Personal Area Networks*), desenvolvida por um consórcio de empresas, das quais se destacam a Intel, Nokia, Ericsson, Toshiba e IBM, que foi definida pelo padrão IEEE 802.15.1. O objetivo de seu desenvolvimento foi a substituição de cabos entre dispositivos portáteis, computadores *desktop* e periféricos em geral [SIG 01].

Características como baixo consumo de energia, elevado grau de miniaturização, alta mobilidade e facilidade de uso fazem do Bluetooth um sistema ideal para ser utilizado em equipamentos portáteis em ambientes de pequeno alcance.

Arquitetura Bluetooth

Segundo Miller em [MIL 01], qualquer dispositivo eletrônico que utilize a tecnologia Bluetooth para comunicação é um dispositivo Bluetooth (Figura 1). Esse é composto de dois módulos. O primeiro chamado *host*, onde se localizam as camadas superiores de aplicação, e o segundo, módulo Bluetooth, que combina todos os componentes (*hardware* ou *software*).

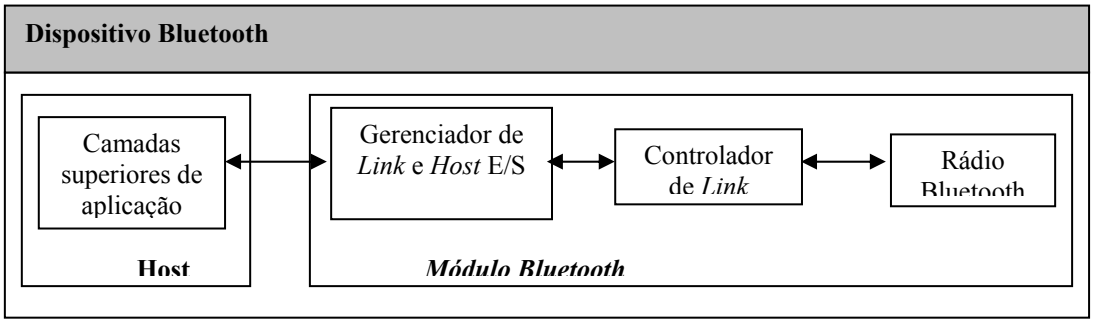

Figura 1: Dispositivo Bluetooth [SIG 01].

Pilha de Protocolos

Diferente dos modelos OSI e TCP/IP, a pilha de protocolos Bluetooth não representa uma pilha de protocolos convencional. Devido ao Bluetooth interoperar com diversos tipos

de dispositivos tais como telefones, *headsets*, celulares e outros, seria difícil categorizar a sua pilha de protocolos de maneira a atender todos os dispositivos que utilizam tal tecnologia.

A Figura 2 exibe a pilha de protocolos Bluetooth – e a Tabela 2 as classifica – dividida em quatro camadas principais: uma de protocolos principais, uma de substituição de cabo, uma de controle de telefonia e a de outros protocolos já existentes.

Tabela 2: Protocolos e camadas da pilha de protocolos Bluetooth [HEL 01].

Camadas de Protocolo	Protocolos
Principais Protocolos Bluetooth	Bluetooth Rádio, Baseband, LMP, L2CAP, SDP
Protocolo de substituição de cabo	RFCOMM
Protocolos de controle de telefonia	TCS Binary, AT-Commands
Protocolos Adotados	PPP, TCP/UDP, IP, OBEX, WAP, vCard, vCal, IrMC, WAE

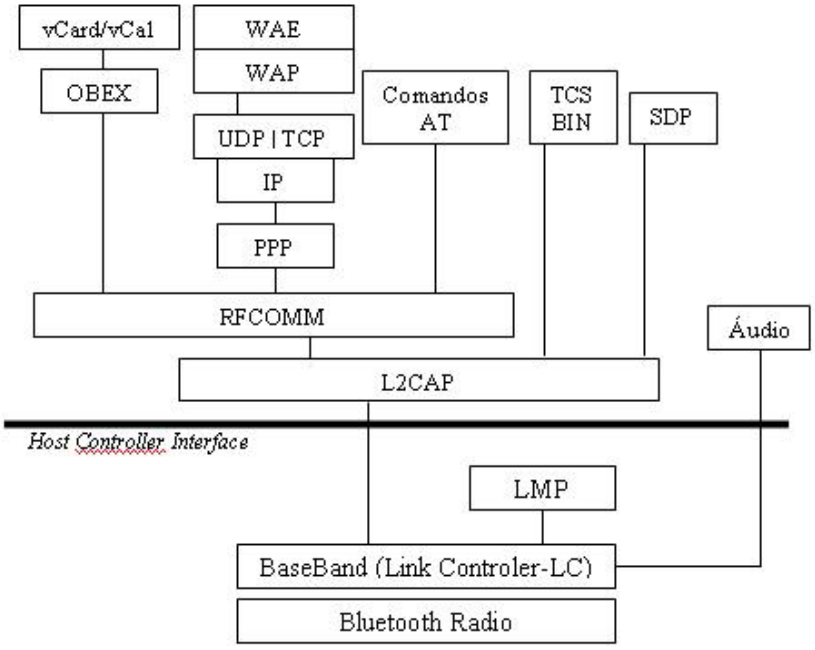

Figura 2: Especificação da Pilha de Protocolo Bluetooth [SIG 01].

Os principais protocolos do Bluetooth são cinco: Bluetooth Rádio, Baseband, LMP, SDP, L2CAP.

Bluetooth Radio

Bluetooth Radio é a camada mais inferior da especificação Bluetooth. Define os requisitos do transceptor (*transceiver*) Bluetooth que opera na banda ISM de 2.4GHz. Bluetooth Radio utiliza as tecnologias FHSS (*frequency hopping spread spectrum*) e TDD (*Time Division Duplexing*) [BLUETOOTH].

A tecnologia FHSS na tecnologia Bluetooth caracteriza-se pelos saltos de freqüência de banda em 79 canais, utilizando uma seqüência de saltos pseudo-aleatórios, a uma taxa de 1600 saltos por segundo, sendo que cada canal é utilizado para transmissão por 625μs entre um salto e outro para um canal diferente [FER 03], como pode ser visto na Figura 3.

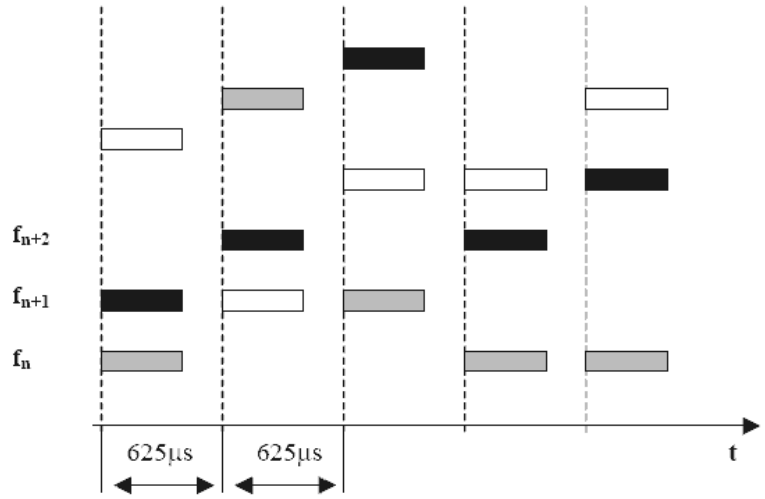

Figura 3: Salto de freqüência por divisão de tempo [AUS 00].

Bluetooth Radio define três classes de potência para os dispositivos Bluetooth como mostra a Tabela 3.

Tabela 3: Classes de potência.

Classe de potência 1	Feita para dispositivos de longo alcance (por volta de 100 m), com uma potência máxima de saída de 20 dBm e 100 mW.
Classe de potência 2	Feita para dispositivos comuns (10 m de alcance), com uma potência de máxima de 4 dBm e 2,5 mW.
Classe de potência 3	Feita para dispositivos de curto alcance (1 m de alcance), com uma potência de máxima de 0 dBm e 1 mW.

A taxa de transferência máxima de *link* alcançada com a versão 1.1 do Bluetooth é de 1 Mbps. Tal taxa é facilmente alcançada através da técnica de modulação GFSK (*Gaussian Frequency Shift Keying*) [BHA 01].

Toda unidade Bluetooth tem um sistema de *clock* interno que determina a temporização e o salto do transceptor. O *clock* do Bluetooth é derivado de um *clock* nativo, o qual nunca é ajustado e desligado. Para sincronização com as outras partes, apenas os *offsets* são usados, adicionados ao *clock* nativo e provendo *clock* temporário, os quais são mutuamente sincronizados [STDJ 01].

A temporização e o salto de freqüência do canal de uma *piconet*[1] são determinados pelo *clock* do nó mestre. Quando a *piconet* é estabelecida, o *clock* do mestre é comunicado aos escravos. Os escravos armazenam o *offset* para ser usado enquanto comunica com um mestre em particular e usa isso para sincronizar o canal. Como o *clock* local não é modificado, diferentes *offsets* podem ser usados para participar de várias piconets [STDJ 01]. A conjunção de duas ou mais piconets dá origem a uma scatternet (Figura 4).

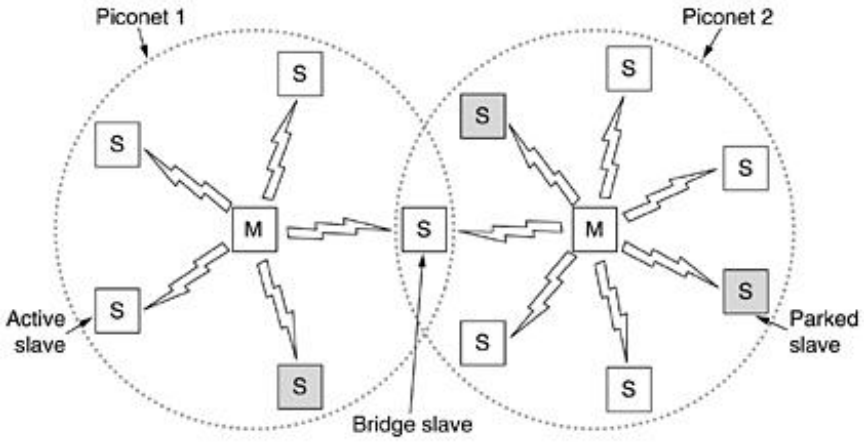

Figura 4: Duas piconets conectadas formando uma scatternet [TAN 03].

Baseband

Baseband permite a conexão de Rádio Freqüência entre dispositivos. Essa camada controla a sincronização de dispositivos Bluetooth e a seqüência de salto de freqüência a ser usada [AUS 00], cuidando também do nível mais baixo de criptografia para segurança das ligações.

Pacotes que viajam através de ligações sem fio são de responsabilidade da banda básica, que fornece dois tipos de ligações físicas. O primeiro, o ACL (*Asynchronous Connectionless*), que suporta tráfego de dados no modo de melhor esforço (*best-effort*), podendo ser dados de controle ou dados do usuário. Ligações ACL suportam conexões simétricas (velocidade de *download* igual a de *upload*) de 433,9 kbps, conexões assimétricas de 723,2 kbps de *download* e 57,6 kbps de *upload*, conexões por comutação de pacotes e conexões *one-to-many* (um para muitos). O segundo tipo é o *Synchronous Connection-Oriented* (SCO), que suporta conexões de dados e voz através de conexões simétricas, *circuit-switched* (comutação de circuito) e ponto-a-ponto. Ligações SCO são geralmente

[1] Rede Bluetooth na qual podem existir até oito dispositivos interligados, sendo um deles o mestre e os outros, dispositivos escravos. *Piconets* independentes e não-sincronizadas podem se sobrepor ou co-existir na mesma área. Nesse caso, forma-se um sistema *ad hoc* disperso denominado *scatternet*, composto de múltiplas redes Piconet, cada uma contendo um número limitado de dispositivos.

utilizadas para transmissão de voz, podendo ocupar três canais de 64 kbps do tipo PCM (*Pulse Code Modulation*) ou CVSD (*Continuous Variable Slope Delta*).

O protocolo *Baseband* permite o uso flexível desses tipos de canais, de modo que diferentes pares de dispositivos numa mesma *piconet* possam utilizar diferentes tipos de canais e o tipo de canal pode variar arbitrariamente durante uma sessão [FER 03]. A largura de banda utilizada em cada canal e a quantidade de banda liberada para cada dispositivo escravo é controlada pelo dispositivo mestre [MIL 01].

Protocolo Gerenciador de Ligação (LMP – Link Manager Protocol)

É responsável por estabelecer o processo através do qual dois dispositivos transferem informações de reconhecimento inicial que permite a configuração e o controle do canal a ser estabelecido entre dois dispositivos [FER 03].

Segundo documentação da Au-system [AUS 00], a camada LMP negocia os tamanhos dos pacotes que serão utilizados durante a transmissão, escolhe e controla os modos de gerenciamento de potência, *hold, sniff e park*, e estados da conexão de um dispositivo em uma *piconet*. A camada cuida também de aspectos de segurança como autenticação entre dispositivos, anexação ou remoção de escravos de uma *piconet* e faz a troca de regras entre mestre e escravo.

Protocolo de Adaptação e Controle de Ligação Lógico (L2CAP – Logical Link Control and Adaptation Protocol)

Faz a adaptação dos protocolos da camada superior para transmissão sobre a banda base (*Baseband*). Portanto, deve ser visto trabalhando em paralelo com o LMP [HEL 01].

Serviços oferecidos para protocolos de camada superior são os de dados orientados a conexão e não-orientados a conexão.

As principais tarefas desse protocolo são:

a) Multiplexação: L2CAP suporta multiplexação de protocolos, uma vez que um grande número de protocolos (SDP, RFCOMM, TCS Binary, etc) pode operar sobre ele simultaneamente [JUN 01].
b) Segmentação e Remontagem: Pacotes de dados excedendo a Unidade Máxima de Transmissão (MTU) devem ser segmentados antes de serem transmitidos e remontados quando recebidos. L2CAP fornece suporte para protocolos e aplicações de mais alto nível transmitirem e receberem pacotes de dados de até 64 Kbits, uma vez que ele suporta datagramas IP.
c) Qualidade de Serviço: O estabelecimento de uma conexão L2CAP permite a troca de informações relativas à Qualidade de Serviço corrente para conexão entre dois dispositivos Bluetooth.
d) Grupos: A especificação do protocolo L2CAP suporta uma abstração de grupo que permite implementações de mapeamento de grupos sobre uma *piconet*.

Acima da L2CAP, existem diferentes camadas, uma delas é a RFCOMM, responsável pela substituição de cabos. A partir dela tem-se o PPP e acima deste as camadas IP e TCP/UDP.

Protocolo de descoberta de serviço (SDP – Service Discovery Protocol)

Divulga serviços do dispositivo, passo necessário anterior a uma conexão entre dois dispositivos. O SDP provê meios para a descoberta de serviços disponíveis quando o dispositivo cliente entra em uma área onde previamente existe um dispositivo Bluetooth operando (mestre). Também fornece funcionalidade para detectar quando um serviço está indisponível.

Protocolo de Emulação de Cabos (RFCOMM)

O RFCOMM (*Radio Frequency Communications Protocol*) emula/substitui uma conexão serial RS-232 (ponto-aponto) sobre a Baseband Bluetooth. Ele fornece capacidades de transporte para camadas de mais alto nível.

Perfis

Perfis Bluetooth foram criados para permitir a interoperabilidade entre dispositivos. É um conjunto de funcionalidades projetadas para dispositivos Bluetooth que descrevem cenários de usuários fazendo uso de comunicação sem fio. Eles são descritos na Tabela 4.

Segundo Tanenbaum em [TAN 03], ao contrário de outras tecnologias como 802.11, que especificam canais de comunicação de protocolos de redes deixando para os projetistas

de aplicações a tarefa de descobrir a utilidade dos canais, a especificação Bluetooth versão 1.1 identifica 13 aplicações específicas e fornece diferentes pilhas de protocolos a cada uma. Entretanto apenas os perfis de Acesso Genérico e de Descoberta de Serviço são obrigatórios em todos os dispositivos.

Tabela 4: Os perfis do Bluetooth, adaptado de [TAN 03].

Perfil	Descrição
Acesso Genérico	Procedimentos para gerenciamento de enlaces
Descoberta de serviço	Protocolo para descobrir serviços oferecidos
Porta Serial	Substitui um cabo de porta Serial
Intercâmbio genérico de objetos	Define o relacionamento cliente/servidor para movimentação de objetos
Acesso LAN	Protocolo entre um computador móvel e uma LAN fixa
Rede *dial-up*	Permite que um *notebook* conecte-se através de um telefone móvel
Fax	Permite que um equipamento de fax móvel comunique-se com um telefone móvel
Telefonia sem fio	Conecta um aparelho telefônico à sua estação-base local
Intercomunicador	Intercomunicação digital
Fone de ouvido	Permite a comunicação de voz sem o uso das mãos
Push de objetos	Fornece um meio para intercambiar objetos simples
Transferência de arquivos	Fornece um recurso mais geral de transferência de arquivos
Sincronização	Permite sincronizar um PDA com outro computador

O perfil de acesso genérico não é realmente uma aplicação, mas sim a base através da qual aplicações serão realmente construídas. Sua principal função é prover e manter um meio seguro para canais de comunicação entre mestre e escravos. O perfil de descobrimento de serviço também é relativamente genérico e é usado por dispositivos para o descobrimento de serviços oferecidos por outros dispositivos. Espera-se que todo dispositivo Bluetooth implemente esses dois perfis. Os demais perfis são opcionais.

Java e Bluetooth

A criação de aplicações em Java pode ser realizada por meio do uso da API Java para Bluetooth (`javax.bluetooth.*`) [ENR 04], definida na Requisição de Especificação Java JSR-082 [JSR-82]. A interação que a mesma tem com as camadas da pilha de protocolos Bluetooth, servindo de interface entre aplicação e pilha Bluetooth, pode ser vista na Figura 5.

A empresa Atinav produz uma das mais utilizadas implementações do JSR-82 com suporte para dispositivos que executam J2ME CLDC, J2ME CDC [CORE] e J2SE [J2SE, HOP 04, ATINAV]. Além da Atinav, outras implementações podem ser utilizadas como a AvetanaBluetooth [AVETANA, AVETANABLUETOOTH] e outras [FIT 05, OBEX4J]. Para os testes realizados neste trabalho, foi utilizada a API AvetanaBluetooth.

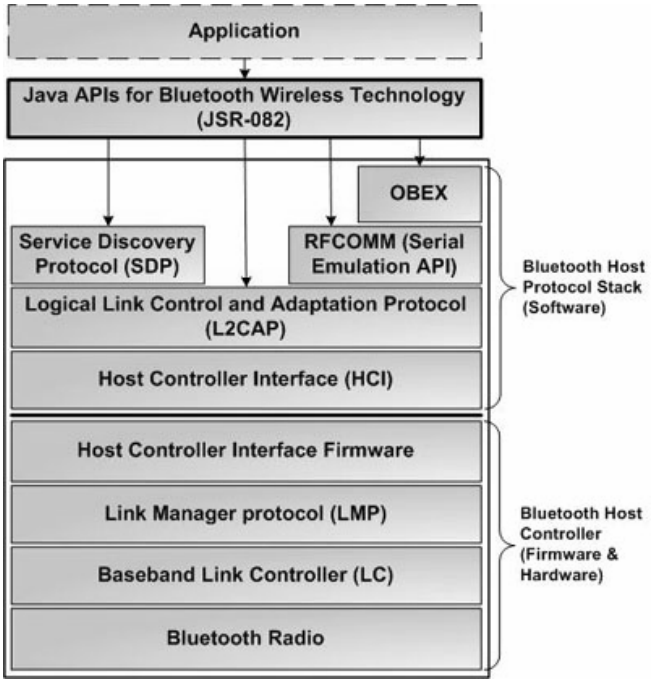

Figura 5: Interação da API Java para Bluetooth com a pilha Bluetooth [ENR 04].

Voltado ao estabelecimento de comunicação através de Bluetooth e Infravermelho entre máquinas virtuais que utilizam APIs J2SE e J2ME, é proposto na JSR-197 [JSR-197] um pacote opcional que implementa um *framework* para conexão genérica capaz de estabelecer comunicação através de tecnologia Bluetooth e Infravermelho entre JVMs que implementam tais APIs.

2.4.1.3 Rede celular

A essência de uma rede celular é o uso de múltiplos transmissores de baixa potência. Devido à escala de tais transmissores ser pequena, uma área pode ser dividida em células, cada uma servida por sua própria antena. Cada célula aloca uma faixa de freqüência e é servida por uma Estação base que consiste de um transmissor, um receptor e por unidade de controle. Células adjacentes suficientemente distantes de outras podem usar a mesma faixa de freqüência dessas [STA 02].

Os principais componentes de um sistema celular são: a supramencionada Estação Base, a Estação Móvel e o Equipamento Central de Processamento ou Centro de Comutação Móvel [ARA 03].

A Estação Móvel tem a função de telefone móvel e utiliza o sistema celular para o estabelecimento de comunicação. É composta de unidade de controle, transceptor e antena para comunicar-se com a Estação Base mais próxima [ARA 03].

A Estação Base realiza a ligação entre Estação Móvel e Equipamento Central de Processamento e é composta de unidade de controle, equipamento de rádio base e antena. Ela é responsável pelo gerenciamento dos canais dentro da célula, supervisiona chamadas, diagnostica problemas, transmite dados entre o Equipamento Central de Processamento e Estações Móveis, monitora a potência do sinal das Estações Móveis, realiza a varredura de quais estações móveis ativas nas células adjacentes e relata a potência para a central, determinando o mapeamento de qual célula deve servir uma dada Estação Móvel quando esta cruza de uma célula para outra, fato conhecido como *handoff* [ARA 03].

O Equipamento Central de Processamento é a central responsável pelas funções de comutação e sinalização para as Estações Móveis localizadas em uma área geográfica designada como a área do MSC. A diferença principal entre um MSC e uma central de comutação fixa é que a MSC considera a mobilidade dos assinantes (locais ou visitantes), inclusive o *handoff* da comunicação quando estes assinantes se movem de uma célula para outra. O MSC liga uma ou mais Estações Base à Rede de Telefonia Pública Comutada.

Além do conceito de *handoff*, há também o de *roaming*. Nele a Estação Móvel encontra-se fora da área normal de serviço. Para que haja conexão nessas condições, rastreia-se a localização do assinante, transfere-se suas informações de sua área normal de serviço para a área visitada e habilita-se o serviço de *roaming* através de comunicação entre as MSCs envolvidas.

A rede celular tem como tecnologias UMTS (*Universal Mobile Telecommunications System*), GPRS (*General Packet Radio Service*), GSM (*Global System for Mobile Communication*s), CDMA (*Code Division Multiple Access*) e WCDMA (*Wideband Code Division Multiple Access*) que estão distribuídas nas diferentes gerações, constituindo a evolução da rede celular de longa distância.

GPRS é um serviço que permite o envio e a recepção de dados através de uma rede de telefonia móvel.

2.5 Dispositivos

Dispositivos para computação ubíqua devem possuir a capacidade de se comunicar, podendo alguns deles fazê-lo mesmo que estejam em movimento. Para isso, devem implementar comunicação de rede sem fio; estes dispositivos são conhecidos como dispositivos móveis.

Um ponto importante nos dispositivos móveis é que tenham autonomia para processar as informações durante a locomoção e, para que possuam tal autonomia, devem dispor de baterias que viabilizem sua utilização durante o período em que são movidos. Apesar do uso de baterias viabilizarem a locomoção dos usuários com seus dispositivos móveis, as baterias também são um fator limitante na utilização desses dispositivos devido ao tempo de operação. Esse fato é considerado não só em dispositivos como PDAs[2], mas também naqueles que funcionam apenas como captação de informação do meio, os sensores. Os últimos possuem características restritas de consumo de bateria ainda mais rigorosas, devido ao seu tamanho reduzido. São instalados de forma ordenada, quando o ambiente é de fácil acesso, ou de forma desordenada, quando o ambiente não é de fácil acesso sendo necessário realizar a instalação através de lançamento a partir de aeronaves. Há também sensores que apresentam mobilidade, estando acoplados a um equipamento móvel, como uma roupa ou PDA, ou a animais para monitoramento.

Com isso, têm-se tanto os dispositivos móveis, como PDAs, sensores e etiquetas de RFid, que acompanham o usuário, quanto os dispositivos fixos, como sensores e dispositivos com sistemas embutidos, que comunicam-se com o ambiente computacional através de uma rede, seja ela cabeada ou sem fio.

Os dispositivos com sistemas embutidos fazem parte do ambiente pervasivo estando presentes em eletrodomésticos – geladeiras, fogões, microondas, etc – e podem utilizar recursos de rede para informar e realizar tarefas para o usuário. Exemplos como o de uma geladeira que faz a leitura das embalagens de produtos munidas de etiquetas de RFid e informa o usuário quais produtos estão em falta, baseado num histórico de compras e na varredura do que há na geladeira, quando ele está fazendo compras. Ou o de uma máquina de lavar roupas que realiza a leitura dos RFids das peças de roupa que informa quais devem ser lavadas separadamente e configura o processo de lavagem adequado àquelas peças. Ou ainda o de um fogão ou microondas que prepara um prato, realizando a leitura do RFid do mesmo e fazendo o *download* da receita deste.

---

[2] *Personal Digital Assistent*

2.5.1 Tipos de dispositivos

Segundo Banavar em [BAN 02], os dispositivos utilizados em computação pervasiva podem ser classificados em: controles inteligentes, utensílios inteligentes, dispositivos de acesso a informação e sistemas de entretenimento.

Controles inteligentes operam no monitoramento e alteração de características do ambiente, seja através de controles locais, como configurações feitas pelo usuário ou pelo próprio sistema de computação ubíqua – baseado em regras extraídas do ambiente –, seja através de controles remotos ou por meio de configurações realizadas pelo usuário que acessa um portal *Web* para fazê-lo. São exemplos de controles inteligentes *smartcards*, controles de linhas de produção, de residências e de automóveis.

Utensílios inteligentes, mais complexos do que os controles inteligentes, podem conter os últimos e implementam o conceito de descentralização, onde cada utensílio trabalha de forma colaborativa para o fornecimento de um serviço. São exemplos de utensílios inteligentes terminais de caixa eletrônico e eletrodomésticos.

Dispositivos de acesso à informação são dispositivos móveis tais como PDAs, telefones celulares e s*martphones*, dispositivos que agregam funcionalidades de PDAs e telefones celulares. São exemplos de dispositivos de acesso à informação celulares e PDAs. Para os testes empregados neste trabalho, foram usados celulares Motorola modelo SVLR L7, como o apresentado na Figura 6.

(a) **(b)** **(c)**

Figura 6: (a) Smartphone Treo 650, (b) PDA LifeDrive e (c) celular L7.

Sistemas de entretenimento constituem os dispositivos que disponibilizam multimídia. São exemplos de tal tipo de sistema aqueles que captam e reproduzem mídias de áudio e/ou vídeo e equipamentos inteligentes de entretenimento. São exemplos de sistemas de entretenimento aparelhos televisores e câmeras digitais.

Dispositivos móveis apresentam restrições de recursos de processamento, memória e armazenamento. Já sensores têm tais restrições ainda mais agravadas devido à capacidade de suas baterias serem mais escassas. Redes de Sensores Sem Fio utilizam protocolos próprios de comunicação em rede que visam o mínimo tráfego de dados para que haja o menor consumo de energia, uma vez que as transmissões de dados em rede sem fio consomem mais energia do que o processamento local. Técnicas de fusão de dados e modos de operação dos transmissores de rádio auxiliam na redução do consumo.

2.5.2 Componentes dos dispositivos

No que se refere aos componentes dos dispositivos móveis, estes possuem limitações de processamento, memória, energia e interface com o usuário e são descritos abaixo segundo Burkhard em [BUR 02].

2.5.2.1 Processadores

Melhorias na fabricação de CMOS têm propiciado estruturas cada vez menores e com um número cada vez maior de transistores. A tensão de operação foi de 3,3V em 1995 para 1,35V em 2002, havendo menor emissão de calor e *caches* maiores. Devido a limitação de capacidade das baterias oferecidas pela tecnologia atual, a tecnologia de processadores tem avançado a ponto de oferecer mecanismos que permitem a redução da tensão do dispositivo através de *software*, reduz o ciclo de *clock*, tornando os processadores mais lentos, mas economizando energia. Processadores como SpeedStep [PROCESSORS] e Centrino [INTEL] da Intel realizam gerenciamento de energia ajustando a freqüência interna do *clock* e tensão do núcleo. Tais processadores adaptam-se a mudanças no fornecimento de energia, desligando partes da CPU que não são necessárias no processamento corrente. A transição entre os modos de operação é transparente.

2.5.2.2 Memória

O desenvolvimento da tecnologia de memórias é orientado, em parte, pelos telefones inteligentes, câmeras digitais, tocadores de MP3 e PDAs. Atualmente, é viável a integração de vários megabytes em dispositivos moveis, através de tecnologias como a dos micro-discos rígidos removíveis (Microdrive da IBM que suporta de 340 MB a 1GB), e memórias *Flash* não voláteis, em *smartphones* e PDAs. A memória RAM dos *smartphones* e PDAs requer

menos energia e provê acesso mais rápido. Com capacidade entre 2 e 23 MB [TREO], contém o Sistema Operacional e dados da aplicação. Tais dispositivos geralmente possuem *slots* de expansão para memória adicional. Tipos de memória disponíveis incluem:

a) RAM estática – SRAM que é ideal para cachê. Requer pouca energia, não realiza *refresh*, possui esquema simplificado de endereçamento e armazena dados que mudam com freqüência, sendo que outros dados são armazenados em *flash* ou DRAM;
b) RAM Unitransitor – Ut-RAM, estática e dinâmica em um único chip, possui maior capacidade de memória em *chips* menores, que requer menos energia;
c) RAM magneto-resistiva – MRAM e RAM ferroelétrica – FRAM, ambas são uma tendência na combinação de memória magnética e semicondutora com comportamento similar ao da RAM estática, que requer menos energia, pois não tem ciclos de regeneração dos bits *(refresh)* que a RAM dinâmica tem. Com estas tecnologias de memória, é necessário apenas um único tipo de memória nos dispositivos móveis, pois elas substituem a combinação de volátil e não volátil atualmente em uso.

2.5.2.3 Telas

Nos dispositivos de acesso a informação, o CRT, Tubo de Raios Catódicos, é substituído pela LCD, Tela de Cristal Liquido. Apesar de ter um custo mais alto, a LCD pesa menos e consome menos energia. As tecnologias de LCD são:

a) OLED – Diodo Orgânico Emissor de Luz, inventado há 15 anos, mas só atualmente disponível comercialmente;
b) Semicondutor Cristalino: compostos orgânicos que permitem a construção de dispositivos flexíveis (construído em qualquer tamanho e cor);
c) LEP - Polímero Emissor de Luz;
d) CoG – Chip sobre vidro;
e) LCoG - Cristal Líquido sobre Vidro. Essas tecnologias permitem a criação de telas minúsculas (tamanho de *pixel* de 10 micrômetros). Como são extremamente pequenas, requerem alguma forma de magnificação (canhão multimídia, capacetes etc).

2.5.2.4 Bateria

Uma tecnologia que limita a velocidade de desenvolvimento de novos serviços para os dispositivos é a bateria. A velocidade de desenvolvimento das baterias é menor do que a

de outras tecnologias. Desta forma, qualquer avanço é rapidamente superado pelo aumento de consumo de energia dos processadores mais rápidos. Pesquisadores do mundo todo trabalham com o objetivo de descobrir novas fontes de energia que sejam mais leves, suportem mais tempo de fala e de processamento de aplicações, além de serem menos agressivas ambientalmente. Baterias de Níquel-Cadmo (NiCad), pesadas e sujeitas a perda de capacidade, foram substituídas por tecnologias como as de Níquel-Metal Híbrido (NiMH), mais leves, maior capacidade e menor agressividade ambiental. Baterias de Lítio-íon (Li ion), encontradas hoje em vários tipos de equipamentos eletrônicos, possuem baixa capacidade de energia, mas suportam um tempo mais longo de fala graças a redução da necessidade de energia dos dispositivos modernos. Células de lítio polímero, que usam gel para o eletrólito, refletem uma das tecnologias mais modernas na área de baterias. Feita de camadas finas e flexíveis pode assumir qualquer forma ou tamanho.

2.6 *Software*

Software para Computação Ubíqua está sujeito às características apresentadas pelos dispositivos onde será executado, portanto, estas devem ser levadas em consideração na concepção da interface com o usuário; utilizar-se de técnicas para a redução de carga de processamento e memória, de consumo de energia e comunicação entre as diferentes tecnologias de conexão entre os dispositivos.

Para tanto, os sistemas operacionais devem implementar o gerenciamento de comunicação em rede, entrada e saída, memória e processamento que atenda às necessidades da computação ubíqua, bem como as aplicações devem tratar de tais características juntamente de uma interface que se adeque ao tipo de interação com o usuário.

Outros fatores envolvidos na criação de sistemas ubíquos são os conceitos de Ciência de Contexto e de Adaptação de Conteúdo. Sistemas ubíquos são desenvolvidos sobre tais conceitos para prover uma melhor interação do usuário com os dispositivos do ambiente. Tais conceitos são detalhados a seguir.

2.6.1 Sistemas operacionais

Da mesma forma que um sistema operacional de PC é responsável pelo gerenciamento de memória, processos, entrada e saída, os sistemas operacionais de dispositivos móveis também o são. Além disso, o sistema operacional oferece maior transparência, disponibilizando uma máquina virtual que oculta a complexidade dos usuários.

Os recursos encontrados em PC são praticamente os mesmos, mudando apenas de tecnologia. O que diferencia o sistema operacional de uma máquina tradicional do sistema

operacional de um dispositivo móvel? Uma das diferenças é a especificidade do dispositivo. O PC é uma máquina de propósito geral, enquanto que o dispositivo móvel é muitas vezes otimizado para realizar muito bem uma ou algumas tarefas. A especialização do dispositivo é um dos aspectos que determina o projeto do sistema operacional do dispositivo [ARA 03].

Abaixo são descritos alguns dos sistemas operacionais para dispositivos móveis.

2.6.1.1 Palm OS

O sistema operacional Palm OS opera sobre um *kernel* preemptivo multitarefa [RHO 99]. Enquanto uma tarefa executa a interface com o usuário, outras podem manipular operações como entrada de dados a partir de botões ou da área de *grafitti* – dispositivo de entrada de dados sensível ao toque, que permite a escrita na tela do PDA ou *smartphone*. A interface com o usuário permite a abertura de somente uma aplicação por vez. Com isso, a aplicação tem controle total da tela. Conseqüentemente, aplicações são executadas em interface mono-usuário e não podem ser *multithreaded*.

Similar a outras plataformas, aplicações executadas na plataforma Palm OS são orientadas a evento. A aplicação busca eventos da fila do evento e despacha-os apropriadamente [EVENT].

Esse sistema operacional opera sobre processadores ARM de 32 bits, tais como Intel Xscale PXA-250 e Motorola MXL. Requer um mínimo de 4 MB de memória Flash ROM e 300KB de RAM para sua execução. Apresenta *drivers* para cartões SD e MMC, 802.11, integração de protocolos de comunicação e aplicações para troca de objetos (*Exchange Manager*), comunicação serial IrDA, USB e Bluetooth, suporte a rede TCP/IP, PPP, OBEX, telefonia e SMS. Tem suporte a criptografia simétrica RC4, função hashing SHA 1 e algoritmo de verificação RSA, por meio do CPM (*Cryptographic Provider Manager*), além de autenticação PAP, CHAP, MS-CHAP v.1 e MS-CHAP v.2 sobre PPP, incluindo autenticação a servidores RADIUS.

Ele dispõe ainda de uma API para desenvolvimento de aplicações nativas em linguagem de programação C/C++, a API 68K. Como otimização dessa API, foi desenvolvido um projeto que consiste na criação de um componente de *software* para conectividade sem fio em dispositivos móveis – Projeto LabPALM[3].

[3] Projeto LabPALM - Projeto para o Desenvolvimento de um Componente de Conectividade para Plataforma Palm, Desenvolvido pelo DC-UFScar em pareceria com a Palm Solutions do Brasil e Celéstica do Brasil S/A, com recursos da Lei de Informática.

O sistema operacional Palm OS pode ser encontrado tanto em dispositivos da Palm One quanto em dispositivos de outros fabricantes (Kyocera, Sony, IBM e Symbol) e em modelos como os *smartphones* Treo [MU 05].

A partir da versão 5.0, o Palm OS passou a suportar processadores ARM [ARM], a ter encriptação de dados, novos *slots* de expansão e comunicação através de rede sem fio Wi-Fi. As versões mais recentes possuem também comunicação via Bluetooth e Infravermelho; hoje encontra-se na versão 5.4 (Garnet).

A versão mais recente do Palm OS, o Cobalt, traz como características um sistema totalmente em 32 bits que é executado sobre processadores ARM, permitindo a inserção de recursos extras e que os aplicativos fiquem mais rápidos. Tem suporte a *multithreading* e *multitasking*, permitindo a execução de aplicativos em *background* e também a definição da prioridade das aplicações. Isso é realizado, protegendo-se a memória e o processo de cada aplicação para evitar conflitos.

No que se refere à comunicação, o sistema que possibilita o acesso via GPRS, Bluetooth, etc. Há também *drivers* para comunicação através de RS-232, IrDA, USB, cartões SD [SDCARD] e MMC [MMCPLUS], TCP e UDP, IP versões 4 e 6, PPP e OBEX.

Apresenta também sistema multimídia com aproximadamente metade do código do sistema operacional Be OS, comprado pela PalmSource em 2001, e o PACE, que simula versões anteriores do Palm OS para que se possa executar aplicações criadas nas mesmas.

2.6.1.2 Linux embarcado

O Linux Embarcado é na verdade um Linux reduzido com suporte para dispositivos móveis. Possui as seguintes características [ARA 03]:

a) Arquitetura microkernel - serviços e características podem ser compilados no núcleo ou carregados como módulos dinamicamente ligados em tempo de execução;
b) Suporte multiusuário;
c) Sistema multitarefa, preemptivo, com escalonadores de tempo-real opcionais;
d) Suporte básico a múltiplos processadores;
e) Tamanho de núcleo podendo variar de 200KB a vários MB (uma crítica ao linux para dispositivos móveis é a de que ele necessita de 2 a 8 MB de memória em tempo de execução);
f) Suporta memória virtual com paginação;
g) Suporta versões reduzidas do X-Window para interface de usuário;
h) Possui versões para processadores MIPS, ARM, Motorola e Intel.

2.6.1.3 Windows CE

Windows CE é um sistema operacional multitarefa, o que permite que seja utilizada uma aplicação enquanto outra é executada em *background* (segundo plano). Como pode ser instalado em diferentes dispositivos, deve ser configurado para cada um deles, o que pode ser realizado por ser oferecido em módulos.

Principais características [ARA 03]:

a) Gerente de Gráficos/Janelas/Eventos – GWE;
b) Suporta Armazenamento de Objetos (arquivos, registros e uma base de dados);
c) Suporta Interfaces de comunicação Infravermelho via IrDA, TCP/IP e *drivers* seriais;
d) Suporta número ilimitado de *threads,* usadas para monitorar eventos assíncronos, tais como: interrupção de *hardware* e atividades do usuário (depende de memória física disponível);
e) Suporta comunicação entre processos – IPC (seções críticas, mutex e eventos);
f) Suporta oito níveis de prioridades de *threads* (do ocioso ao tempo crítico). As prioridades não podem mudar, exceto quando uma *thread* de baixa prioridade retém recurso que outra de alta prioridade precisa;
g) Suporta Interrupções que são usadas para notificar o SO (Sistema Operacional) sobre eventos externos;
h) Atende requisitos de um S.O. de tempo real não-crítico para aplicações do tipo: comutação de telefonia, controle de processos de fabricação, sistemas automotivos de navegação, etc.
i) Trata os vários tipos de armazenamento da mesma forma: cartões de memória *flash*, *drives* de disco, ROM, RAM;
j) Suporta até 32 MB por processo.

2.6.1.4 TinyOS

TinyOS foi desenvolvido pelo Departamento de Engenharia Elétrica e Ciência da Computação (EECS) da Universidade da Califórnia em Berkeley para trabalhar com o microprocessador ATMEL AT4. Trata-se de um sistema operacional muito simples e compacto, baseado em eventos, desenvolvido para atender alguns dos requisitos das RSSFs, as quais são operações intensivas de concorrência com requisitos mínimos de *hardware* e para a economia de energia. A linguagem de programação no TinyOS é uma linguagem C estilizada chamada NesC.

Ele é composto por um SO simples, um ambiente de desenvolvimento com código aberto, um modelo e uma linguagem de programação. O SO utiliza orientação a eventos e um conjunto de serviços.

Apresenta um escalonador de tarefas, que é uma fila (FIFO – *First In First Out*), utilizando uma estrutura de dados de tamanho limitado. O escalonador, desenvolvido para concorrência intensiva é não-preemptivo e não tem mecanismos sofisticados como fila de prioridades. Como os nós sensores – dispositivos onde é utilizado tal SO – monitoram eventos, podendo ocorrer vários destes ao mesmo tempo, as tarefas que vão atender esses eventos têm que executar eficientemente e o SO tem que ser projetado para que todos os eventos possam ser atendidos em tempo hábil. Devido aos recursos dos nós sensores serem extremamente limitados, o TinyOS também foi construído para trabalhar com poucos recursos e facilitar o desenvolvimento de componentes de *software* voltados à eficiência e modularidade.

Seu conjunto de ferramentas permite o desenvolvimento de código para esse sistema. O código fonte é aberto e pode ser baixado em [TINYOS].

Ele apresenta também um modelo de eventos que permite ter concorrência utilizando pouco espaço de memória. Um modelo baseado em troca de contexto e pilha iria requerer que o espaço de pilha fosse reservado para cada troca de contexto, além de requerer processamento em cada troca. Além disso, a energia é um recurso precioso. O modelo baseado em eventos alcança um ganho de 12 vezes comparado com o modelo de troca de contexto. Os ciclos não utilizados da CPU fazem com que o nó entre em um estado de "*sleep*", ao invés de procurar ativamente um evento.

TinyOS também apresenta um conjunto de serviços, interfaces e componentes que implementam os principais serviços de uma aplicação de RSSF que estão disponíveis junto com o sistema. Estes serviços são:

a) Rádio, MAC, troca de mensagens, roteamento: componentes que implementam a pilha de protocolos do TinyOS;
b) Interface de sensores: conjunto de interfaces para os mais variados tipos de sensores que podem ser utilizados em um nó sensor com o TinyOS.
c) Gerência de energia: energia é o recurso mais crítico de uma RSSF.
d) Depuração: provê componentes e ferramentas que permitem a depuração de código.
e) Temporizadores: provê componentes de acesso aos relógios do nó sensor.

Tudo isso faz com que o TinyOS tenha como objetivos:

- Atender sistemas embutidos em redes: o sistema deve “dormir”, mas permanecer vigilante a estímulos. Quando um ou mais eventos ocorressem, estes deveriam ser atendidos;
- Acompanhar os avanços tecnológicos: prevendo um desenvolvimento da tecnologia de MEMS, o TinyOS deve acompanhar os avanços tecnológicos, para que não fique ultrapassado rapidamente. A tendência dos circuitos integrados é de continuar a reduzir as dimensões de tamanho, ficando mais barato, e incluindo tecnologias de pouco consumo de energia (*low power*).

TinyOS é voltado especificamente para a arquitetura Mica Motes. Tarefas e componentes de energia, sensoriamento, computação e comunicação dessa plataforma são implementados no TinyOS.

2.6.2 Linguagens de programação para dispositivos móveis.

Algumas linguagens de programação vêm sendo utilizadas no desenvolvimento de sistemas de computação ubíqua e Java é uma que se sobressai devido à sua grande flexibilidade por ser multiplataforma. XML é outra linguagem que facilita a troca de dados através da Internet.

2.6.2.1 Java

A linguagem de programação Java faz parte de uma tecnologia, na qual há uma plataforma do tipo *middleware* onde são executadas as aplicações implementadas nessa linguagem. As aplicações são compiladas em um formato especial, o *bytecode*, que é interpretado pelo *middleware*, a Máquina Virtual Java (MVJ), permitindo que aplicações desenvolvidas numa dada plataforma de sistema operacional sejam executadas em várias outras, tais como Linux, Windows, Machintosh, PalmOS, dentre outras, constituindo sua característica de linguagem multiplataforma. Há MVJs para diferentes plataformas tanto de SO quanto de *hardware*, permitindo-se que aplicações sejam desenvolvidas para serem executadas em PCs, PDAs, *smartphones*, etc.

Além da MVJ, a plataforma Java apresenta um conjunto de APIs que facilitam o desenvolvimento das aplicações devido àquelas implementarem muitas das funcionalidades necessárias ao desenvolvimento de aplicações, acelerando a implementação dessas.

A plataforma Java é distribuída em três edições, a J2EE (*Java 2 Enterprise Edition*) [J2EE], J2SE (*Java 2 Standard Edition*) e J2ME (*Java 2 Micro Edition*), cada uma delas com APIs que oferecem funcionalidades necessárias à área de aplicação a qual se destina (Figura 7). A J2EE destina-se a aplicações com propósito de atender a aplicações servidoras,

implementadas em múltiplas camadas. J2SE destina-se a aplicações utilizadas em estações de trabalho (*desktops*) como PCs e *notebooks*. J2ME é voltada a aplicações em dispositivos móveis, como PDAs, celulares, sistemas embarcados, *smartphones* e *smartcards* e terá importância neste trabalho, pois o mesmo apresenta resultados através de estudos de caso em dispositivos móveis acessando arquitetura JAMP.

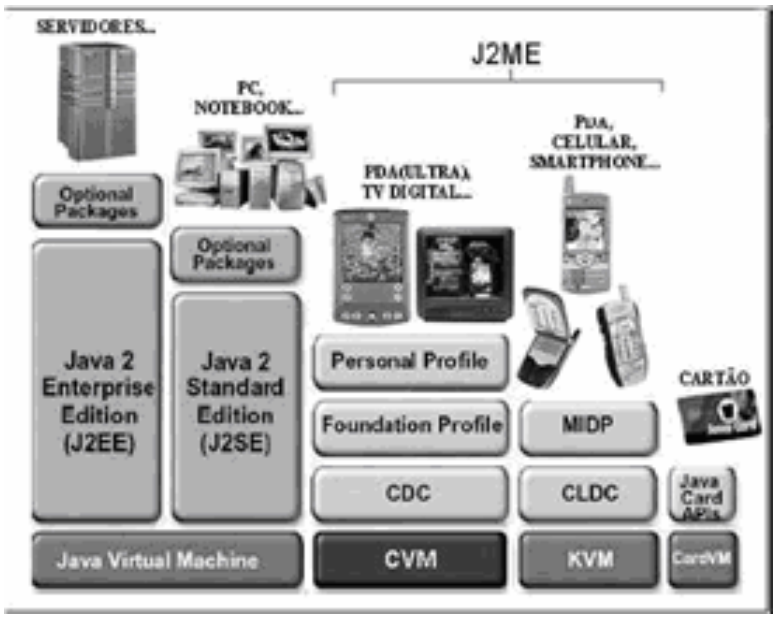

Figura 7: J2ME frente às demais edições Java [FMJ 04].

J2ME utiliza os conceitos de Configuração e Perfil.

Configuração descreve características e bibliotecas para a MVJ, que é utilizada em dispositivos com limitações de tela, memória e processamento. São definidas duas configurações, CDC (*Connected Device Configuration*), que se destina a dispositivos com capacidade de memória superior a 512KB [CONNECTEDa] e usa a MVJ CVM – ocupando um espaço de memória de 2MB para instalação –, podendo ser instalada em sistemas automotivos e PDAs avançados [FRES 01], e CLDC (*Connected Limited Device Configuration*), que destina-se a dispositivos com memória entre 160KB e 512KB [CONNECTEDb] e opera sobre máquina virtual KVM – podendo ser instalada na maioria dos dispositivos, como celulares, *smartphones* e PDAs, por exigir pouco espaço de memória para instalação. Para que celulares possam executar a configuração CLDC, estes devem apresentar requisitos de *hardware* de no mínimo 160 KB de memória para a MVJ, processador de 16 bits, no mínimo, de baixo consumo de energia e conexão de rede *wireless* de 9,6 kbps, 144 kbps ou 2 Mbps. E como requisitos de *software* devem ter suporte a um

subconjunto da MVJ e da linguagem, a fim de permitir o desenvolvimento de aplicações móveis [LAMONT].

Configurações não são suficientes para determinar o que deve conter nas APIs a ponto de satisfazer as necessidades das aplicações em cada tipo de dispositivo, pois estes podem variar de placas, ou mesmo cartões inteligentes com SOs embutidos, até PDAs e *smartphones* com grande poder de processamento, diferentes funcionalidades e tipos de conexão. Com isso, houve a necessidade de se definir um mínimo de funcionalidades às diferentes faixas de dispositivos com características semelhantes de *hardware*, tais como capacidade de processamento, memória e conectividade. Por isso, é necessário algo mais, além das configurações, para classificar os dispositivos e isso é obtido através dos Perfis.

Sobre as Configurações, localizam-se os Perfis de dispositivo. Alguns deles são MIDP (*Mobile Information Device Profile*), RMI *Profile*, *Foundation Profile*, PDA *Profile*, *PersonalJava*, etc e a disposição de um Perfil pode ser vista na Figura 8.

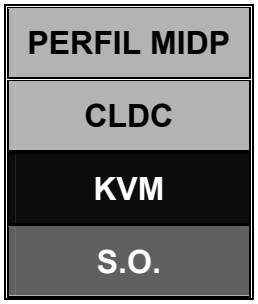

Figura 8: Aplicação na arquitetura J2ME.

O desenvolvimento de aplicações para J2ME pode ser realizado por meio do uso de ferramentas como o J2ME *Wireless Toolkit*. O J2ME *Wireless Toolkit* é acompanhado de um emulador que permite a seleção do dispositivo (PDA ou celular) a ser emulado, além de exemplos com código fonte.

Aplicações baseadas na tecnologia J2ME são chamadas de MIDlets [GETTING] e estendem a classe MIDlet.

2.6.2.2 XML

Linguagem de Marcação Estendida (XML – *eXtensible Markup Language*) é um padrão de processamento de documentos proposto pelo W3C (*World Wide Web Consortium*), mesmo grupo responsável pela supervisão do padrão HTML. XML é uma meta linguagem que permite a criação e formatação de documentos através da criação de rótulos (*tags)* definidos pelo desenvolvedor. Essas *tags* podem ser aplicadas em um ou mais documentos [XML 99], tendo também um importante papel na transferência de uma ampla variedade de dados na Web [XML 05]. Ele é um subconjunto de SGML (*Standard Generalized Markup*

Language), XML facilita a implementação e interoperabilidade entre documentos SGML [ISO] e HTML.

Documentos baseados em XML descrevem uma classe de objetos de dados e parcialmente descrevem o comportamento dos programas que os processam. Eles são compostos de unidades do armazenamento chamadas de entidades, que contêm dados gramaticalmente analisados ou não. Os dados analisados gramaticalmente são compostos dos caracteres, alguns representando dados e outros, formatação (*tags*). As *tags* descrevem as estruturas lógicas de armazenamento dos documentos.

Um processador XML é usado para ler documentos XML e fornecer acesso a seu conteúdo e estrutura. Sendo assim, o processador XML realiza esse trabalho para a aplicação. Ou seja, o comportamento requerido a um processador XML diz respeito à leitura dos dados XML e retorno da informação que é fornecida à aplicação.

2.6.3 Técnicas para a obtenção da Ubiqüidade

Outras técnicas relacionadas à Computação Ubíqua são a Ciência de Contexto e a Adaptação de Conteúdo, descritas a seguir.

2.6.3.1 Ciência de Contexto

Consiste na captação de informações do ambiente, através de sensores, para prover a interação entre usuários e dispositivos pervasivos de sistemas ubíquos do ambiente. É pautada em cinco questionamentos [WEI 91]: Quem? O que? Onde? Quando? Por quê?

Os sistemas que implementam ciência de contexto fazem a leitura de informações do ambiente, bem como dos usuários que se encontram nele, e criam regras de comportamento baseadas nas atividades desses usuários. Tais regras são construídas a partir de configurações feitas pelo usuário e também através do aprendizado do sistema, que altera, cria ou exclui regras existentes no mesmo, baseado nos acontecimentos ocorridos no ambiente [SAL 99].

2.6.3.2 Adaptação de Conteúdo

O termo adaptação de conteúdo refere-se à modificação da representação de objetos ou conteúdos, com o intuito de encontrar uma versão ideal desses conteúdos. Por exemplo, sua adequação às capacidades do dispositivo de acesso (processador, memória, *display*, etc), às restrições de transmissão impostas pela rede de acesso e às características do usuário que acessa tal conteúdo [CLA 05].

2.7 Descobrimento de recursos

Na Computação Pervasiva, a interação entre dispositivos e sensores é imprescindível para que se obtenha a funcionalidade desejada do ambiente. Nela, notificação e descobrimento de dispositivos permitem que o espaço pervasivo mude e evolua dinamicamente sem uma reengenharia do sistema [HEL 02].

Para a resolução de problemas de configuração automática no estabelecimento de conexão e cooperação em redes locais, são empregados protocolos de descobrimento de serviços. Alguns deles são: Jini, Universal Plug and Play [UPNP], Salutation [SALUTATION], Service Location Protocol (SLP) [GUTa 99, OPENSLP] e Bluetooth SDP [BET 00].

Um computador móvel não é capaz de usar uma impressora próxima sem que tenha o *driver* da mesma; talvez um PDA obterá um acesso lento a Internet por não saber qual o servidor *proxy caching* mais próximo. Enquanto a computação móvel evolui além da habilidade de se estabelecer uma conexão sem fio para ler e-mail ou navegar em qualquer rede e com qualquer dispositivo, está limitada a explorar recursos, dispositivos e serviços locais [HEL 02]. Ou seja, há uma limitação para buscar e utilizar recursos que se situam além dos limites de uma rede local. Com isso, o *framework* desenvolvido neste trabalho colabora para o avanço do descobrimento de serviços por dispositivos móveis através da Internet a partir de um dispositivo Bluetooth.

A seguir, seguem alguns protocolos para descobrimento de serviço para uso em redes locais.

2.7.1 Jini

Jini é um ambiente distribuído, desenvolvido sobre RMI, que constitui uma arquitetura para construção de sistemas distribuídos. Ele permite que aplicações utilizem serviços de rede sem o conhecimento de quais protocolos estão sendo utilizados [JINI].

A tecnologia utiliza recursos tais como serialização [JINI] de objetos, *sockets* e RMI. Esta tecnologia traz para um nível ainda maior o conceito de plugabilidade intercomponentes, sejam eles componentes de *hardware* ou de *software*.

Utilizando Jini, impressoras, câmaras de vídeo, PDAs, etc, podem se conectar diretamente à rede. A grande vantagem é que os equipamentos já existentes na rede saberão que o novo dispositivo foi adicionado e encontra-se disponível.

Jini trabalha com três componentes principais: Serviços, Clientes e Localizador de Serviços. Exemplos de Serviços são as impressoras, câmaras de vídeo e disco rígido. Clientes

utilizam os serviços Jini disponíveis e o Localizador de Serviços atua como um *broker*, entre Serviços e Clientes [JINI].

Um serviço Jini é definido através de sua interface declarada na linguagem Java. Quando este é iniciado na rede, ele próprio exporta seu objeto que implementa a interface de serviço. Clientes localizam o serviço através das interfaces que ele oferece. Quando um cliente invoca um serviço, ele recebe um código correspondente para a comunicação com aquele serviço. O desenvolvedor implementa o serviço escolhendo qual o tipo de comunicação: RMI, CORBA ou algum protocolo em particular [WAL 99].

Jini não é totalmente transparente ao programador uma vez que toda uma série de passos deve ser executada para que se possa utilizar a infra-estrutura.

Com Jini, clientes deixam de se preocupar em manter os *drivers* de suporte aos dispositivos para manterem um conjunto de interfaces que descrevem os dispositivos e serviços de *software* disponíveis na rede. Conforme haja uma evolução da tecnologia, conjuntos padronizados de interfaces devem ser especificados trazendo todos os benefícios de plugabilidade esperados. O cliente passa as características que necessita, escolhe o serviço e o executa [UERJ].

2.7.1.1 Tecnologias no mercado

Quando lançado em 1999, Jini foi introduzido em produtos Epson, Canon, Seagate e Quantum. Porém, no final do mesmo ano, deixou de fazer parte dos produtos devido a tais fabricantes acreditarem que a tecnologia se tornaria instável num futuro próximo. Por outro lado, PsiNaptic, empresa líder em computação pervasiva, desenvolveu o Jmatos, produto com tecnologia Jini. Ele suporta Tiny Internet Interface, conhecida como Tini, a qual tem uma MVJ embarcada. Tini é um pequeno micro-controlador com um grande conjunto de interfaces *onboard* [HEL 02].

Diferente de Jini, UPnP integra produtos de outros fabricantes além de máquinas que executam Windows XP, tais como OSGi/UPnP Gatespace e MetroLink, que transportam o padrão OSGi num ambiente pervasivo [OSGI], Home Director, um sistema de rede doméstica baseado em X10 para controle de dispositivos domésticos acessados via web ou por PDAs e roteador AnyPoint Home da Intel [ANYPOINT]. Há também SDKs (*Software Development Kit*) que permitem a implementação do suporte ao UPnP em produtos Allegro, Virata, Intel, Lantronix, Atinav, Microsoft e Siemens [UPNP]. Tais SDKs estão disponíveis para plataformas Windows, Windows CE e Linux e suportam C, C++ e Java.

Muitos produtos utilizando Salutation estão disponíveis no mercado, mas esses são produtos para automação de escritório para utilização em fax, impressoras, copiadoras e *scanners*. O NuOffice da IBM é um sistema de escritório em rede baseado em Lotus Notes que permite aos usuários importar e exportar documentos para qualquer dispositivo Salutation [NUOFFICE].

Com o aval da IETF e alinhado com outros protocolos como LDAP, DNS, e DHCP, o SLP [BET 00] tem sido bastante aceito por desenvolvedores como um protocolo de descobrimento de serviço simples e mínimo. SLP é utilizado em impressoras e produtos de rede Lexmark, Xerox, Minolta, IBM e Novel.

2.7.2 UPnP

O UPnP auxilia no anúncio, descobrimento e controle de dispositivos de rede e serviços. É uma extensão do modelo Microsoft *Plug-and-Play* para periféricos e com ele um dispositivo pode adentrar dinamicamente a uma rede, obtendo endereço IP, informando suas capacidades à rede e obtendo informações sobre a presença e os serviços oferecidos por outros dispositivos.

A pilha de protocolos utilizada para descobrimento de serviços, anúncio, descrição e para eventos é exibida na Figura 9.

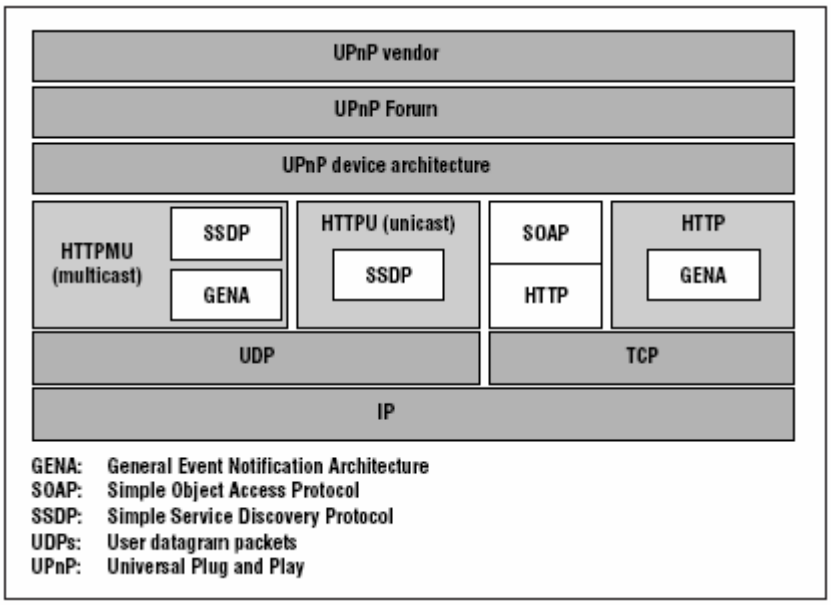

Figura 9: Pilha de protocolos do Universal Plug and Play [HEL 02].

Para o descobrimento de serviços, o UPnP utiliza o SSDP (*Simple Service Discovery Protocol*) que anuncia a presença de dispositivos ou serviços.

UPnP usa XML para descrever características e capacidades de dispositivos. Tal descrição encontra-se presente em um arquivo XML na rede. Diferente de Jini, a descrição de

serviço do UPnP é mais poderosa devido a documentos XML terem tal capacidade de representação dos dispositivos.

Após o descobrimento do dispositivo por um ponto de controle, esse obtém mais informações sobre o uso, o controle e a coordenação desse dispositivo a partir do arquivo XML do mesmo. O controle é expresso como uma coleção de objetos SOAP (*Simple Object Access Protocol*) [SOAP] e das URLs (*Universal Resource Locator*) presentes no arquivo XML. A descrição de um serviço inclui uma lista de ações para as quais os serviços respondem e uma lista de variáveis que modelam o estado do serviço em tempo de execução [HEL 02].

2.7.3 Salutation

Voltado ao descobrimento de serviço e uso de dispositivos e serviços para diferentes fins, o consórcio Salutation [PAS 01] vem desenvolvendo outro padrão, chamado Salutation. Sua arquitetura provê um método padrão para dispositivos, serviços e aplicações descreverem e anunciarem suas características entre si. A arquitetura possibilita a pesquisa e descobrimento baseado em características particulares [HEL 02].

A arquitetura é composta por dois componentes principais, o gerenciador Salutation e o gerenciador de transporte. O primeiro é a base da arquitetura e similar ao serviço de *lookup* do Jini ou o ponto de controle do UPnP, sendo melhor definido como um *broker* onde provedores de serviços registram suas características. Quando um cliente requisita um serviço ao seu gerenciador Salutation local, todos os gerenciadores realizam a consulta e então o cliente poderá utilizar o serviço retornado. O gerenciador Salutation localiza-se acima dos gerenciadores de transporte que provêem canais de comunicação seguros.

Os serviços disponibilizados pelo Salutation são o de registro, descobrimento, disponibilidade, gerenciamento de sessão e o Salutation-Lite, uma versão reduzida da arquitetura Salutation para dispositivos com poucos recursos.

2.7.3.1 Serviço de registro

O gerenciador Salutation contém um *registry* para manter informações sobre os serviços; um cliente pode efetuar ou desfazer seu próprio registro. Todos os registros são feitos com o gerenciador Salutation local ou com aquele mais próximo do cliente.

2.7.3.2 Serviço de descobrimento

O gerenciador Salutation descobre serviços presentes em outros gerenciadores e os registra. Serviços remotos são descobertos através de tipos e conjuntos de atributos

especificados pelo gerenciador local. A cooperação entre gerenciadores forma um serviço de *lookup* conceitualmente semelhante ao do Jini.

2.7.3.3 Serviço de disponibilidade

É um serviço que consiste na verificação periódica da disponibilidade de um serviço, executada pelo gerenciador a pedido do cliente. Tal serviço implementa uma versão mais rudimentar daquilo que ocorre nos serviços de eventos do Jini e do UPnP.

2.7.3.4 Serviço de gerenciamento de sessão

Opera em um dos seguintes modos: nativo, emulado ou *salutation*. No modo nativo, as trocas de mensagens ocorrem através de um protocolo nativo, sem a participação do gerenciador. No modo emulado, o gerenciador transmite as mensagens entre clientes e serviços sem inspecionar o conteúdo das mesmas. No modo salutation, além de transmitir as mensagens, os gerenciadores definem o formato a ser utilizado na sessão.

2.7.3.5 Salutation-Lite

Voltado para dispositivos de pequeno porte, tal como PDAs, e para comunicação de rede com pouca largura de banda, tais como Infravermelho e Bluetooth.

2.7.3.6 Service Location Protocol

É um padrão do IETF (*Internet Engineering Task Force*), voltado ao descobrimento de serviço descentralizado, leve e extensível. Os serviços são definidos por meio de URLs que descrevem o tipo de serviço e o endereço para um serviço particular. Por exemplo, **`service:printer:lpr://hostname`** é uma URL para um serviço de impressora disponível em *hostname*. Baseado no serviço de URL, usuários e serviços podem navegar pelos serviços disponíveis no domínio, escolher e utilizar aqueles que quiserem [GUTb 99].

Existem três agentes no SLP, o UA (usuário), o SA (serviço) e o DA (diretório). O UA é uma entidade de *software* que envia requisições de serviços de interesse de uma aplicação de usuário. O SA envia um *broadcast* anunciando o interesse de um serviço. Como um serviço de repositório centralizado de informações, o DA armazena anúncios dos SAs e processa as requisições de descobrimento das UAs.

2.7.4 UDDI

O *Universal Descripton, Discovery, and Integration* (UDDI) é um serviço de registro de *Web Services* [WER, 03]. Ele serve para descrever, descobrir e integrar *Web Services* e faz isso fornecendo um diretório de busca para os serviços. É como um *broker* onde os serviços são registrados por servidores e localizados por clientes.

O UDDI tem uma especificação que define um serviço de registro para *Web Services*, com:

a) APIs SOAP [W3Ca 04] utilizadas na publicação e obtenção de informações de registros UDDI;
b) Esquemas XML do modelo de dados do registro e do formato das mensagens SOAP;
c) Definições *Web Services Description Language* (WSDL) [W3Cb 03] das APIs SOAP;
d) Definições de registro UDDI de diversos sistemas de identificação e categorização, que podem ser utilizados para identificar e categorizar registros UDDI.

O registro de serviços do UDDI consiste em três componentes:

a) **Páginas brancas**: endereço, contato e identificadores conhecidos;
b) **Páginas amarelas**: categorizações industriais baseadas em taxonomias padrão;
c) **Páginas verdes**: informação técnica sobre serviços publicados por empresas.

UDDI é projetado para ser consultado por mensagens SOAP e para prover acesso a documentos WSDL que descrevem o protocolo de tradução e formatos de mensagens necessários para interagir com serviços Web listados em seu diretório. Sua estrutura é exibida na Figura 10.

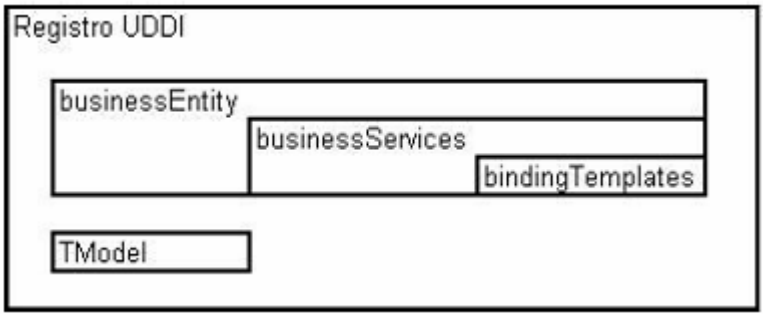

Figura 10: Estrutura de um registro UDDI.

2.7.5 Bluetooth SDP

Diferente de Jini, UPnP, Salutation ou SLP, o Bluetooth SDP pode ser utilizado especificamente em dispositivos Bluetooth. Voltado para o problema de descobrimento de serviço, não provê acesso a serviços, gerenciamento de serviços, anúncio de serviços ou registro de serviços nem notificação de eventos quando esses se tornam indisponíveis [HEL 02].

SDP suporta pesquisa por classe de serviços, pesquisa por atributos de serviços e serviço de *browsing*. O último é utilizado quando um cliente Bluetooth não tem prioridade de conhecimento dos serviços disponíveis em sua vizinhança. SDP é estruturado como um perfil Bluetooth e executa no canal de conexão orientada da camada de ligação lógica L2CAP. Salutation propôs um mapeamento entre seu descobrimento de serviço e o Bluetooth SDP.

Tal mapeamento é sinérgico porque complementa Bluetooth adicionando serviço de anúncio, *broker* e eventos.

2.8 Desafios

A emergência da computação ubíqua proverá uma rica e motivadora oportunidade para pesquisas futuras. Atualmente encontra-se num estágio pouco adiantado de desenvolvimento. Conseqüentemente, estudá-la envolve estudar algo que ainda não existe. Pesquisadores nesse campo ainda estão idealizando e criando problemas tanto quanto estão resolvendo-os e teorizando sobre seus efeitos. É necessário encontrar maneiras de manter o rigor da pesquisa científica sem restringir sua habilidade de imaginar e, além disso, encontrar maneiras de estudar assuntos pessoais em um nível global. Por exemplo, como computadores trajáveis [BAS 97] interagem com os ambientes por onde passam e como esses afetam a satisfação dos indivíduos ou como sua produtividade deve ser estudada no contexto da difusão global da tecnologia móvel. Finalmente, a pesquisa em computação ubíqua precisa transpor as barreiras tradicionais entre o social e o técnico bem como os níveis de análise individual, de grupo e de organizações [LYY 02b].

Características referentes aos componentes dos dispositivos, tais como autonomia de bateria e tamanho de tela, e também relativas à capacidade do sinal e taxa de transmissão de dados em rede são desafios a serem vencidos por dispositivos móveis. As limitações dos dispositivos provocam uma grande demanda pelo estudo da Adaptação de Conteúdo, baseado não só nas características do usuário, no que diz respeito ao formato do conteúdo que se deseja ter nos dispositivos, mas também relacionado a qual conteúdo o dispositivo é capaz de exibir segundo seu tamanho de tela, taxa de transmissão e perenidade do sinal. Por exemplo, o sistema poderia deixar de realizar o *download* de figuras quando a taxa de transmissão de rede, no dispositivo, diminui.

Baseado no contexto do usuário, a infra-estrutura de *software* provida pelo sistema ubíquo deve ser capaz de encontrar, adaptar e prover aplicações adequadas ao ambiente computacional do usuário [BAN 02]. Tais requisitos sinalizam a necessidade de um ambiente onde as entidades capazes de oferecer serviços sejam disponibilizadas a clientes, sejam eles humanos ou outras aplicações, como serviços, e não como fragmentos de *hardware* dispersos na rede. Para isso, este trabalho colabora para o adicionamento tanto de dispositivos móveis como de clientes, permitindo que estes acessem os serviços de forma transparente através de interfaces bem definidas.

2.9 Conclusões

À medida que novos conceitos surjam a partir das necessidades de mobilidade de usuário, as demais mobilidades – de dispositivos e serviços – serão necessárias para atender os requisitos de utilização de serviços em qualquer lugar e a qualquer hora. Diante disso, a Computação Ubíqua e a Pervasiva devem atender às esses requisitos, oferecendo formas de comunicação e processamento de informações remotas, mantendo a conectividade do usuário com os serviços que este acessa. Para isso, tecnologias de comunicação de dados devem sofrer mudanças para permitir a utilização de recursos e também, o surgimento de novos dispositivos estarão providos dessas tecnologias.

Para que tudo isso ocorra, estudos de interconectividade, onde diferentes tecnologias de rede possam ser utilizadas para o consumo de serviços remotos, deve levar em consideração o que os dispositivos oferecem e como as tecnologias podem se entrelaçar para que os serviços sejam utilizados. Dessa forma, Sistemas Operacionais, dispositivos móveis, tecnologias de rede, protocolos, como descobrimento de serviços, e outros aspectos, como Ciência de Contexto e Adaptação de Conteúdo, foram discutidos neste capítulo.

Ainda há muito que ser desenvolvido no âmbito do descobrimento de serviço. Muitos dos produtos, conforme se pode observar na literatura, não implementam requisitos especiais e mobilidade, tornando incerta sua utilização em ambientes móveis e pervasivos. Daí o interesse deste trabalho.

SDPs são projetados para o funcionamento em LANs. Jini, por exemplo, tem sua utilização limitada ao IP *multicast*, tornando-se inadequado à utilização por clientes móveis que requisitam serviços em WANs. Outro problema é a falta de suporte para dispositivos móveis, pois, por exemplo, Jini tem sua utilização impedida por requerer MVJ e RMI no lado cliente. Tal problema virá a ser sanado com o advento de Jini Surrogate [SURROGATE]. Utilizando Jini Surrogate, dispositivos que não possuem MVJ – sem capacidade para baixar e executar código Java – ou não tratam RMI, poderão usar *proxies* na LAN com tal comportamento.

Há ainda uma grande carência no que tange a informações de contexto por parte dos descobridores de serviço, sem suporte de roteamento, distância do cliente ao serviço, tempo de resposta, carga de serviço e qualidade do mesmo, embora se tenha trabalhado a respeito de questões de mobilidade e contexto para descobrimento de serviço [HEL 00, HEL 03a, HEL 03b, NOR 00].

No âmbito deste trabalho, dois tipos de descobrimentos são feitos: o de recursos (dispositivos) e de serviços Bluetooth, e o de serviços JAMP.

O descobrimento de dispositivos e serviços Bluetooth é realizado por meio do uso do Protocolo de Descobrimento de Serviços (SDP), que localiza dispositivos que dispõem de interface de comunicação Bluetooth e serviços oferecidos nestes dispositivos. Já a localização de serviços na JAMP é obtida através do processo de *trading*, onde um cliente daquele serviço o requisita para o JBroker. Este informa onde o serviço requisitado se localiza, dando uma referência do mesmo para acesso direto por parte do cliente daquele serviço. Com isso, os pontos de acesso Bluetooth para serviços da JAMP, funcionam como um *proxy* aos dispositivos móveis.

Uma aplicação de Computação Ubíqua pode ser suportada por uma infra-estrutura de *middleware*, onde diferentes aspectos do sistema são implementados nessa camada. Nesse sentido, a JAMP enquadra-se como ambiente para proporcionar estudos em Computação Ubíqua por ser um *middleware* onde aspectos de *bind* (localização de serviços) e controle de sessão são garantidos pelo JBroker.

3 MIDDLEWARE

Esse capítulo apresenta um como os *middlewares* são classificados na literatura, situando-os dentro do contexto de Sistemas Distribuídos. A JAMP é apresentada dentro dessa classificação e apresentada a sua estrutura.

Dá-se o nome de Sistema Distribuído ao conjunto de computadores que executa aplicativos de forma cooperativa, de modo que pareça ser uma única entidade computacional [TAN 02]. A implementação de um ambiente pervasivo onde é possível executar sistemas ubíquos pode ser obtida através de um modelo de Sistema Distribuído. São tipos de Sistemas Distribuídos, SOs distribuídos, SOs de rede e sistemas baseados em *middlewares*.

Middleware é uma camada intermediária, localizada entre SO e aplicações, oferecendo funcionalidades a aplicações que sobre elas são executadas, tornando o SO transparente às aplicações, constituindo assim numa infra-estrutura para a implementação de sistemas distribuídos, garantindo heterogeneidade tanto de *hardware* quanto de *software* [TAN 02].

Middleware tem como principais funções, comunicação em rede, sincronização, confiabilidade, escalabilidade e homogeneidade em ambientes heterogêneos [SCH 95, EMM 00, KRA 03], as quais são apresentadas a seguir.

Comunicação de rede

Middlewares utilizam como base a camada de transporte e implementam funcionalidades encontradas nas camadas de sessão e apresentação do modelo ISO/OSI. Eles devem também oferecer uma forma parametrizada para a requisição de serviços de componentes remotos. Isso permitirá que componentes distribuídos em diferentes dispositivos estabeleçam uma forma de comunicação, permitindo que eles trabalhem em conjunto.

Sincronização

Quando há a execução concorrente de componentes que se comunicam, é necessário que haja sincronização entre eles.

A sincronização ocorre de três formas: síncrona, síncrona postergada e assíncrona. Na síncrona, ocorre o bloqueio de um componente enquanto um componente aguarda a conclusão de um serviço requisitado por ele a outro componente. Na síncrona postergada, o componente que requisita um serviço não tem seu processamento interrompido, sincronizando-se no futuro com o componente que atende sua requisição. Já na assíncrona, o componente que atende a requisição, sincroniza-se com o requerente.

Há também casos onde as requisições ocorrem em grupo. Nesse caso, o componente deve sincronizá-las.

Confiabilidade

Há vários graus de confiabilidade entre os protocolos de rede sobre os quais os *middlewares* são construídos, que podem ou não garantir a entrega de requisições feitas a eles.

Maior confiabilidade reflete em queda de desempenho em função da sobrecarga agregada ao protocolo. Portanto, é necessário que haja rigor por parte dos desenvolvedores na escolha dos protocolos a ponto de balancear-se confiabilidade e eficiência.

Segundo Tanenbaum em [TAN 02], há quatro níveis de confiabilidade: melhor-esforço, no-máximo-uma, pelo-menos-uma e exatamente-uma.

A confiabilidade de melhor-esforço não dá qualquer garantia da entrega, processamento ou mesmo de resposta daqueles que recebem as requisições e, portanto, deveria executá-las. No caso de no-máximo-uma, há a garantia, pelo *middleware*, de que a entrega da requisição no destino ocorrerá apenas uma vez. No caso da requisição não ser recebida, aquele que a enviou receberá uma notificação. Na confiabilidade de nível pelo-menos-uma, o *middleware* garante a entrega pelos menos uma vez, podendo repeti-la se necessário. O maior grau de confiabilidade ocorre no nível exatamente-uma, onde se pode garantir que a requisição será entregue uma e somente uma vez ao destino.

Há também outras garantias de confiabilidade como a em grupo e a de *time-out*. Na primeira, chamada de confiabilidade-k, garante-se que pelo menos k componentes receberão a requisição. A segunda define um intervalo de tempo máximo após o qual novas tentativas de entrega de requisições não poderão ser realizadas.

Escalabilidade

Escalabilidade é definida como a capacidade que sistemas têm de aumentar ou diminuir para suportar carga de trabalho aplicada a ele sem sofrer esforço significativo [COU 00, TAN 02].

Emmerich em [EMM 00] define que a dificuldade de se construir um Sistema Distribuído está na possibilidade de se atribuir componentes a dispositivos sem se alterar a arquitetura do sistema ou mesmo o projeto e código dos componentes. Segundo Coulouris e Tanenbaum em [COU 00, TAN 02], isso só pode ser obtido através de níveis de transparência, para os quais a ISO define oito [OPEN], dos quais quatro se aplicam ao contexto de *middleware*: acesso, localização, migração e replicação.

Transparência de acesso refere-se à forma homogênea de acesso a componentes, sejam eles locais ou remotos. Essa transparência reflete na despreocupação com a localização, o que garante uma transparência de localização, sendo que a mudança de localização de um componente, durante o acesso a outro, não deve influir na execução da requisição, havendo transparência de migração. A transparência de replicação consiste em ignorar se há duas ou mais instâncias de um dado componente para o qual um outro componente faz requisições.

Homogeneidade em ambientes heterogêneos

A heterogeneidade pode ocorrer em plataformas de *hardware*, SOs ou em linguagens de programação. A heterogeneidade de SO é resolvida através da utilização de *middlewares*, *brokers* e *proxies* que ocultam, de certa maneira, suas diferenças.

A necessidade de se criar *middlewares* veio da utilização da programação distribuída que requer um conjunto de funcionalidades específicas para resolver questões de comunicação entre aplicações sem que isso seja o foco do sistema implementado. Tais funcionalidades criadas para um determinado sistema, por não estarem de acordo com um padrão, não se aplicam a outros sistemas, tendo que ser reimplementadas sempre que houver a necessidade de utilizá-las. Para que essas funcionalidades sejam disponibilizadas numa camada padronizada sem a necessidade de reimplementação, elas são realizadas na forma de *middleware*.

Na década de 90, os *middlewares* eram utilizados para estabelecer a interconexão entre aplicações e BDs (Bancos de Dados), servindo de ponte entre interfaces de programação, linguagens de manipulação de dados, como SQL (*Structured Query Language*), e protocolos de rede [DEU 94, FIN 95]. Depois, passaram a conter

funcionalidades que propiciaram a construção de componentes [KIN 92]. A partir de tais funcionalidades, é possível que computadores individuais, redes e sistemas de arquivos sejam ignorados, constituindo o conceito de transparência [COU 00] necessário ao desenvolvimento de aplicações distribuídas, obtido através de *middleware*.

Tecnologias de *middleware* para suporte a comunicação foram propostas com o objetivo de fornecer a infra-estrutura necessária para facilitar o desenvolvimento de aplicações distribuídas [COS 02], tratando questões como heterogeneidade de plataforma no que diz respeito a diferenças de *hardware*, SO e linguagem de programação, podendo ainda prover um conjunto de serviços padrão que são típicos de aplicações distribuídas tais como serviço de diretórios e criptografia [MAT 03].

Tecnologias bem estabelecidas como comunicação interprocessos e invocação remota, serviço de nomes, criptografia, sistemas de arquivos distribuídos, replicação de dados e mecanismos de transação distribuída provê infra-estrutura de suporte às aplicações de rede atuais [COU 00].

O modelo dominante ainda é a arquitetura tradicional cliente-servidor, entretanto, o desenvolvimento de aplicações para sistemas distribuídos tem rumado cada vez mais para o uso de *frameworks*, como CORBA, que oferecem um nível mais alto de abstração tal como objetos distribuídos e serviços, incluindo comunicação segura, autenticação e armazenamento persistente [MAT 03]. *Frameworks* de aplicações distribuídas deverão suportar código móvel, fluxo de dados multimídia, mobilidade de dispositivos e usuários e redes espontâneas [COU 00]. Um exemplo de *framework* com suporte a mobilidade de dispositivos e acesso sem fio é o wCORBA [BLA 01].

Sobre os *middlewares*, podem ser construídos os *frameworks*, conjuntos de programas que atendem a necessidades de uma classe de aplicações. Com isso, sobre os *middlewares*, e conseqüentemente sobre os *frameworks*, as aplicações são implementadas pelos desenvolvedores, havendo a preocupação somente com problemas que serão resolvidos pelas mesmas. Ou seja, detalhes como interfaces, protocolos e serviços padronizados são implementados na camada do *middleware*.

Além disso, é papel do *middleware* representar dados de forma homogênea, pois o faz para diferentes plataformas de SO, e gerenciar o estado de comunicação (sessão) entre entidades [BAC 03], o que, no TCP/IP, encontram-se nas camadas de transporte e aplicação [CER 74]. Ele também trata/abstrai questões de rede, sincronização, confiabilidade, escalabilidade e heterogeneidade, tirando do desenvolvedor tais responsabilidades [SCH 95, EMM 00, KRA 93].

3.1 Tipos de *middleware*

Existem na literatura algumas classificações para *middlewares* baseadas nas primitivas oferecidas pelos mesmos. Para Schreiber em [SCH 95] os *middlewares* são divididos em TP (*Transaction Process* - Monitores de Processamento de Transação), RPC (*Remote Procedure Call* - Chamadas Remotas de Procedimentos), MOM (*Message Oriented Monitor* – Monitores Orientados a Mensagens) e ORB (*Object Request Broker* - Intermediadores de Requisições a Objetos). Emmerich em [EMM 00] segue a mesma classificação de Schreiber, porém Mascolo, Capra e Emmerich em [MAS 02] e Myerson em [MYE 02] concordam com tal classificação, exceto com relação a ORB, que para esses é uma evolução de RPC.

Middlewares TP oferecem ferramentas e ambientes para execução de transações distribuídas. Aqueles de RPC tratam da distribuição de lógica de programação através da distribuição de procedimentos, iniciados por um processo e executado por outro de forma transparente. MOMs realizam comunicação assíncrona e são indicados para troca de dados entre aplicações. ORBs permitem a distribuição e compartilhamento de objetos em redes heterogêneas. A plataforma utilizada neste trabalho, JAMP, é uma plataforma de *middleware* classificada como ORB por servir como mediador para a publicação de serviços, por parte dos servidores, e para a consulta dos serviços publicados, por parte dos aplicativos clientes.

Além da classificação que trata da distribuição lógica de aplicação, há também outras formas de classificação que consideram carga de processamento, comunicação adotada e representação de contexto, definidas por Mascolo, Capra e Emmerich em [MAS 02]. No texto de Mascolo, Capra e Emmerich, a classificação por carga tem duas categorias, a *heavy-weight* e a *light-weight*. A primeira trata de *middlewares* com grande necessidade de recursos, podendo oferecer replicação, tratar de aspectos de escalabilidade e sincronização. A segunda categoria necessita de menos recursos.

Na classificação de comunicação de Mascolo, Capra e Emmerich, pode-se dividir os *middlewares* em síncronos e assíncronos, respeitando as mesmas considerações apresentadas no início deste capítulo.

A última forma de classificação relaciona-se à transparência do *middleware*. Se esse é capaz de passar o contexto de execução à aplicação. Ou seja, caso apresente fatores de comunicação, processamento, etc, ele é dito ciente e quando não informa a aplicação a respeito dessas características, é dito transparente. *Middlewares* reflexivos são cientes devido a terem de informar às aplicações a respeito dos aspectos envolvidos em sua execução –

contexto de execução – para que haja interação entre *middleware* e aplicação a ponto da última adaptar-se ao ambiente.

Há outra forma de classificação descrita por Bacon e Harris em [BAC 03], onde *middlewares* TPs e RPCs são considerados serviços de SO, sendo considerados como *middlewares* apenas os orientados a objetos e os orientados a mensagens.

Plataformas de *middleware* para ambientes móveis devem prover meios para lidar com questões de mobilidade e conectividade, as quais não são tratadas por plataformas convencionais.

3.2 JAMP - *Java Architecture for Media Processing*

O desenvolvimento da JAMP teve início em 1996, quando foi criado o JBroker, um *broker* para a mediação entre servidores e clientes de conteúdo multimídia em sistemas distribuídos, na publicação e consulta de serviços distribuídos. Com base no JBroker, serviços foram implementados por outros alunos de mestrado sob orientação do Prof. Dr. Luis Carlos Trevelin e alguns trabalhos foram publicados em eventos da comunidade cientifica. É com base nessa arquitetura que a extensão, proposta neste trabalho, foi desenvolvida.

A construção de novas aplicações desenvolvidas sobre objetos distribuídos requer o desenvolvimento de novas tecnologias, pois necessitam um gerenciamento eficiente, recuperação e disseminação do conjunto heterogêneo de componentes multimídia e documentos. Com a finalidade de suprir tais necessidades, a plataforma JAMP (*Java Architecture for Media Processing*) foi desenvolvida [TRE 98] e caracteriza-se como um *middleware* do tipo ORB.

Um sistema distribuído gerencia recursos [TAN 03] dispersos em uma rede de computadores de forma transparente para seus usuários. O importante nesses sistemas é que os recursos sejam usados através de uma interface uniforme e independente de sua localização. Em comparação aos sistemas centralizados (*mainframes*), a alternativa distribuída apresenta vantagens de menor custo, maior flexibilidade e confiabilidade.

A JAMP consiste de um ambiente para desenvolvimento de aplicações multimídia cooperativas em ambiente distribuído, provê servidores e *frameworks* para suporte a vários formatos de mídias e codificações, assim como processamento e reprodução dentro de aplicações Java ou *applets* [FER 98]. Compõe-se de servidores, um *broker* e um conjunto de *frameworks* para desenvolvimento de aplicações distribuídas. É construída com o paradigma

de orientação a objetos e baseia-se na linguagem de programação Java e em sua API RMI [FAR 98].

Sua organização segue o modelo proposto pela OMG, dividida em Objetos da Aplicação, Mecanismo de Distribuição de Objetos, Objetos de Serviço e *Frameworks* de Domínio.

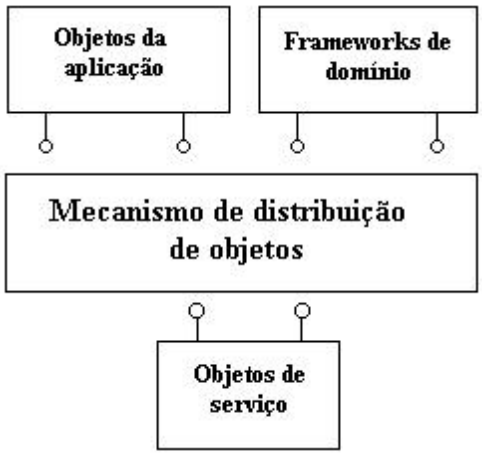

Figura 11: Arquitetura JAMP.

As estruturas exibidas na Figura 11 são assim representadas:

a) **Objetos de Aplicação** referem-se às aplicações existentes que utilizam a JAMP como uma plataforma de distribuição;
b) **Mecanismo de Distribuição de Objetos** utilizam Java RMI como principal forma de distribuição e controle de objetos;
c) ***Frameworks* de Domínio** representam os domínios de aplicação (Áudio, Vídeo, Rope, Midi, Trabalho Colaborativo, Comunicação de Grupo e Comércio Eletrônico), os quais podem ser observados na Figura 13;
d) **Objetos de Serviço** representam a estrutura imprescindível para o funcionamento da JAMP, o JBroker, que atua na arquitetura realizando o processo de *trading* [BAT 04], o qual é exibido na Figura 12.

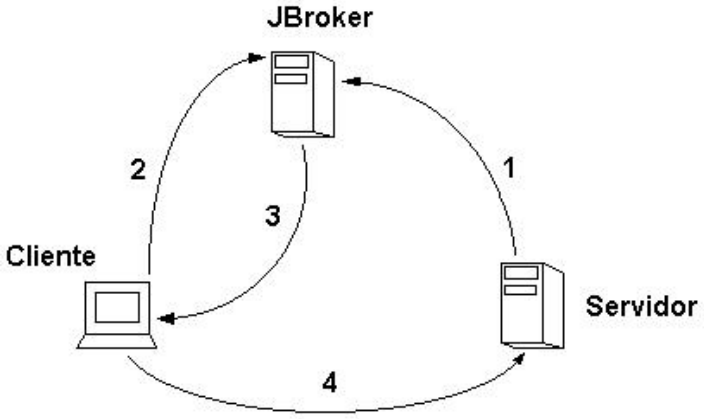

Figura 12: Processo de Trading. (1) Publicação do serviço. (2) Consulta ao JBroker. (3) Referência do objeto de serviço. (4) Acesso direto ao serviço.

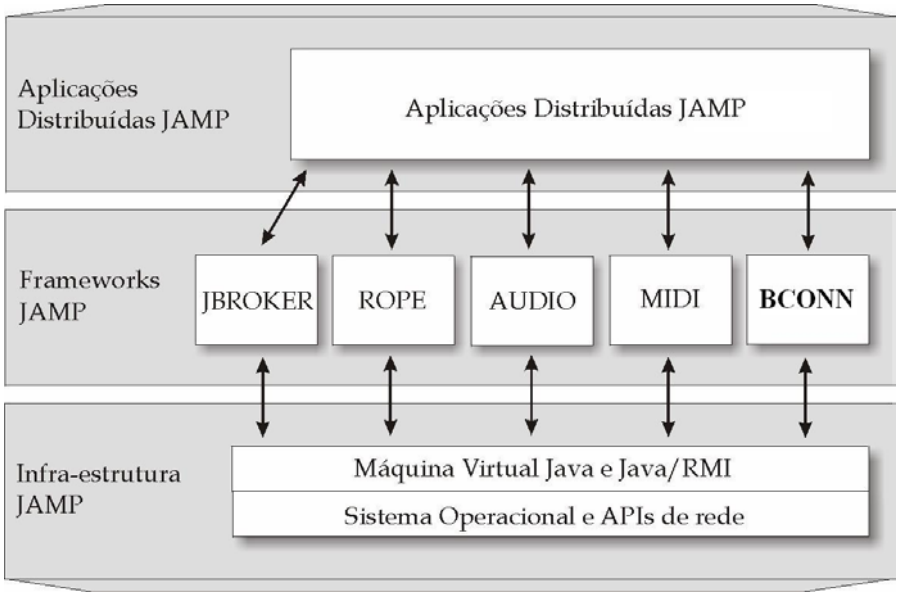

Figura 13: Arquitetura JAMP e suas camadas.

O JBroker é um objeto de serviço que oferece importantes funcionalidades à plataforma. Contém uma base de dados dos servidores disponíveis no momento e permite que os clientes encontrem os servidores distribuídos (objetos remotos) na rede [FMJ 04].

As aplicações multimídia distribuídas da JAMP [MED 99] podem usar todos os *frameworks* de domínio disponíveis na plataforma. Os serviços dos *frameworks* são embutidos nas aplicações através de sua compilação. As aplicações podem ser: Aplicações Java, que são executadas pela MVJ, ou *Applets*, que são executadas pelos navegadores ou *Applet viewers* que executam a MVJ.

O *framework* JavaBroker oferece importantes funcionalidades para a plataforma, provendo acesso ao Serviço de Nomeação e ao servidor JBroker [SOU 01].

Os *frameworks* de domínio, oferecem APIs que são usadas pelas aplicações para incorporar as necessidades de processamento de mídias, trabalho cooperativo e permitir a comunicação distribuída. Nesta camada nota-se a flexibilidade que a plataforma JAMP oferece às aplicações para Web, pois novos *frameworks* podem ser criados conforme a necessidade, ou seja, a plataforma não está restrita apenas aos *frameworks* citados [FER 98]. Com isso, um *framework* para a interconexão de dispositivos em ambientes ubíquos pode ser construído sobre a plataforma, aproveitando-se de toda a infra-estrutura que ela provê.

Alguns dos serviços implementados através de *frameworks* na JAMP são o Serviço de Nomeação e Balanceamento de Carga, o Serviço de Negociação, o Serviço de Notificação, o Serviço de Segurança e o Serviço de Avaliação e Acompanhamento.

O *framework* de domínio de Nomeação e Balanceamento de Carga permite a localização de outros objetos dentro do ambiente distribuído por meio de nomes associados a cada um deles. É possível também a criação de hierarquias e grupos de objetos, permitindo a utilização de métodos de balanceamento de carga em cada grupo [SOU 01].

Através da utilização do Serviço de Negociação, clientes da plataforma JAMP podem localizar objetos dentro da rede através de suas características exportadas ao Trader. Algumas características são definidas pelo usuário e outras são definidas pelo próprio Trader em tempo de execução, permitindo dessa maneira uma consulta mais detalhada. Este serviço complementa as deficiências do Serviço de Nomeação [SOU 01].

O objetivo do Serviço de Notificação é permitir a difusão de mensagens de forma assíncrona dentro de um ambiente distribuído, permitindo que objetos remotos sejam notificados da ocorrência de determinados eventos em outros objetos remotos. Este serviço implementa o modelo *push* da especificação CORBA do OMG [SOU 01]. No modelo *push* o consumidor apresenta uma interface chamada PushConsumer e o fornecedor, uma interface chamada PushSupplier. O fornecedor, quando necessita emitir um evento, ativa a operação de *push* do PushConsumer.

No Serviço de Segurança, a comunicação entre objetos pode ser definida de modo a permitir a criptografia das informações. A autenticação de clientes e servidores, bem como a transmissão codificada entre os objetos, é realizada pelo protocolo SSL (*Secure Socket Layer*) [SOU 01].

3.3 *Middlewares* para interconexão de dispositivos em ambientes ubíquos

Várias questões estão envolvidas no que se refere à interconexão de dispositivos móveis. Transparência de plataforma é interessante para que códigos gerados em determinada

plataforma possam ser executados em outras. Comunicação de rede deve permitir interação entre diferentes dispositivos, móveis ou fixos, em suas diferentes tecnologias. Acesso a serviços disponíveis nessa rede deve ser oferecido de modo fácil a clientes que os utilizam.

Atualmente, transparência de plataforma de *hardware* e sistema operacional pode ser obtida por meio da utilização da linguagem de programação Java, uma vez que um código gerado um determinado SO instalado em certa plataforma de *hardware* pode ser executado em SO e *hardware* diferente – isso em função da flexibilidade que Java oferece, por ser executado sobre uma máquina virtual auto-contida, que oferece sempre a mesma forma de interação com SO e *hardware* em qualquer dispositivo onde possa ser instalada. Facilitação no acesso a serviços disponíveis numa rede pode ser obtida por meio de utilização de APIs para descobrimento de serviços, por exemplo a API Jini, presente na linguagem Java, porém, somente para protocolos baseados em TCP/IP.

A agregação de dispositivos que operam em tecnologias de rede diferentes de TCP/IP viabilizaria a utilização de serviços oferecidos por meio de Jini a tais dispositivos. A partir de tal agregação, serviços poderiam ser disponibilizados para uma gama maior de dispositivos.

A proposta deste trabalho vem a oferecer os serviços já implementados na plataforma JAMP a dispositivos móveis. Com a utilização de APIs para descobrimento de serviços, a arquitetura JAMP poderá prover serviços para dispositivos como PDAs, *smartphones*, etc e a utilização dos mesmos em conjunto com outros dispositivos, compondo uma rede de serviços mais ampla. E isso ainda estabelecendo comunicação não só através de tecnologia de rede TCP/IP, mas também através de Bluetooth.

3.3.1 Trabalhos relacionados

Centralized Wireless Local Area Networks (CWLANs) [GER 02] é uma arquitetura para interconexão de *scatternets* através de um *backbone* UMTS e foi criada com o objetivo de prover acesso Internet a usuários de dispositivos móveis que dispõem de interface Bluetooth. Uma unidade contendo interfaces UMTS e Bluetooth conecta-se à estação base UMTS TDD (*Universal Mobile Telecommunication System, Time Division Duplex*).

RCBEE (*Remotely Controlled Bluetooth Enabled Environment*) [CHA 04] é um ambiente que implementa o controle remoto de vários dispositivos com interface de comunicação Bluetooth dispersos em ambientes domésticos. Estes se comunicam a partir da formação de uma scartternet, onde um de seus nós mestres faz conexão à Internet para prover o controle dos dispositivos através de uma página Web.

Outro trabalho, desenvolvido com objetivo de oferecer serviços de *stream* de áudio com suporte a qualidade de serviço a clientes Bluetooth, é o ubiQoS [BEL 04]. Ele é um *middleware* baseado em Java que interconecta piconets através de pontos de acesso Wi-Fi/Bluetooth. É composto do *shadow proxy*, que trabalha associado ao dispositivo Bluetooth, na fronteira entre a PAN (*Personal Area Network*) e a Internet, e adaptadores de QoS, os chamados Places, que modelam tipicamente nós e podem ser agrupados em domínios que correspondem a LANs (*Local Area Networks*). *Shadow proxy* e adaptadores de QoS são agentes móveis desenvolvidos na plataforma SOMA (disponível em **`lia.deis.unibo.it/Research/SOMA`**) [BEL 04].

O *framework* BConn, desenvolvido neste trabalho, por outro lado, oferece o acesso a serviços já implementados na JAMP, oferecidos por PCs. Permite também a exploração de características de baixo consumo de energia com Bluetooth em redes sem fio de curto alcance para o acesso de serviços remotos, comparado às redes Wi-Fi [LOP 06]. Isso ocorre devido ao Bluetooth operar a uma potência de sinal de transmissão inferior a de redes Wi-Fi.

4 O FRAMEWORK BCONN

Nesse capítulo é descrito o cenário de operação do BConn, seu funcionamento, quais ferramentas foram utilizadas na modelagem do mesmo e o modelo no qual foi baseado.

O desenvolvimento do BConn tem por objetivo estender a arquitetura de *middleware* JAMP, permitindo o acesso de dispositivos móveis a serviços da mesma. Nesse sentido, foi projetado e desenvolvido um *framework* de conectividade nesta plataforma, o BConn. A inovação trazida pelo *framework* reside na possibilidade de ter demais tecnologias não baseadas no TCP/IP fazendo acesso a serviços da JAMP, sendo que a implementação, testes e validação foi feita somente com a tecnologia de rede Bluetooth.

A plataforma JAMP serve de base para a criação do *framework* que provê a rede de serviços implementados na plataforma para dispositivos móveis. A localização de serviços oferecidos é disponibilizada através do uso do *framework* de Negociação, implementada na Plataforma JAMP. Através da sinergia entre serviços nas diversas tecnologias de comunicação em dispositivos móveis, como, por exemplo, o Bluetooth, e JAMP, por meio da infra-estrutura criada no *framework*, serviços da JAMP são oferecidos a usuários utilizando dispositivos móveis.

Isso permite que dispositivos, que antes não podiam consumir serviços da JAMP, devido a esta utilizar RMI como forma de publicação/localização de serviços sobre TCP/IP, passem a fazê-lo. Isso torna os dispositivos móveis parte de uma rede de serviços podendo se comunicar por meio de tecnologias de rede sem fio. E esse acesso a serviços publicados em RMI só é possível devido ao uso de Computação de Borda (*Edge Computing*) [COMPUTING] nos limites da rede TCP/IP, fazendo a ponte entre dispositivos que utilizam essa tecnologia de rede e os dispositivos móveis, que dispõem de outras tecnologias. Tal ponte é realizada pelos chamados pontos de acesso.

Também a possibilidade do uso de uma rede de dados em telefonia celular foi considerada, servindo de base para o acesso de dispositivos móveis em movimento a altas velocidades, como no tráfego em rodovias ou ferrovias. Para tanto, pontos de acesso à rede de serviços da JAMP podem ser implementados como *proxies* GPRS. E nesse caso, tais

pontos de acesso devem apresentar uma configuração do tipo CDC (*Connected Device Configuration*), pois esse apresenta as condições para a instalação do pacote adicional RMI disponível para uso em máquina virtual da versão J2ME. Dispositivos CDC apresentam maior poder de processamento.

Em suma, o *framework* vincula tecnologias de redes sem fio ao ambiente distribuído da JAMP. Nesse sentido, permite que dispositivos móveis possam acessar serviços implementados em quaisquer outros nós que fazem parte da JAMP.

Estudos relacionados vêm sendo desenvolvidos por Chen et al. [CHE 02] para prover *sockets* RMI diretamente sobre a camada L2CAP de Bluetooth. Diferente do trabalho de Chen, mas ainda unindo Bluetooth a RMI, este trabalho permite que dispositivos que dispõem apenas de interface de rede Bluetooth possam acessar serviços suportados pela tecnologia RMI.

O que JAMP oferece de vantagem frente à implementação de Chen é o JBroker, que otimiza o processo de negociação (*Trading Process*), fazendo com que a publicação e localização de serviços seja mais rápida e transparente.

4.1 Cenário de operação do *framework*

O cenário de operação de BConn tem como base os pontos de acesso. O ponto de acesso consiste de um *Personal Computer* (PC) ou celular GPRS onde é executado o serviço de conectividade provido pelo *framework.* Esse ponto de acesso apresenta interfaces de rede sem fio e TCP/IP, ou de telefonia móvel (celular), comunicando-se tanto com os dispositivos móveis presentes na área de alcance, quanto com a Internet ou rede de dados de telefonia móvel (GPRS), oferecendo conectividade com serviços registrados no JBroker. Dessa forma, a busca e utilização de serviços da JAMP podem ser realizadas também a partir de uma rede sem fio.

O ponto de acesso, onde BConn é executado, funciona como um *proxy* para onde são enviadas as requisições de serviços feitas por dispositivos móveis próximos do mesmo. Nele, as requisições geram novas requisições, via RMI, a serviços oferecidos por servidores ou *desktops* em PCs, onde são executados serviços na JAMP. A Figura 14. exibe algumas formas de acesso à JAMP por dispositivos móveis. Dentre elas o acesso de um Tablet a partir de um ponto de acesso implementado em um celular, o acesso de um *laptop* e um PDA a partir de um ponto de acesso em um PC e o acesso de um PDA a partir de um ponto de acesso em um *smartphone*.

A partir do processo de *trading*, oferecido por JAMP, um serviço é localizado pelo JBroker que, em seguida, disponibiliza ao BConn uma referência de onde o serviço é oferecido. De posse dessa informação, o cliente faz uso do serviço, realizando chamadas aos métodos presentes no objeto servidor, como se este estivesse sendo executado na mesma MVJ do ponto de acesso – *proxy* do dispositivo móvel. Isso se deve ao fato da JAMP ser baseada em RMI, API Java que oferece invocação remota de métodos.

Mesmo fazendo uso de RMI, JAMP não apresenta qualquer problema de utilização em redes protegidas por *firewall*, uma vez que técnicas para tunelamento da comunicação RMI – como RMI sobre portas abertas, tunelamento de HTTP para porta e tunelamento de HTTP para CGI/Servlets, todos descritos em [JUR 04] – podem ser utilizadas, como é apresentado por Juric.

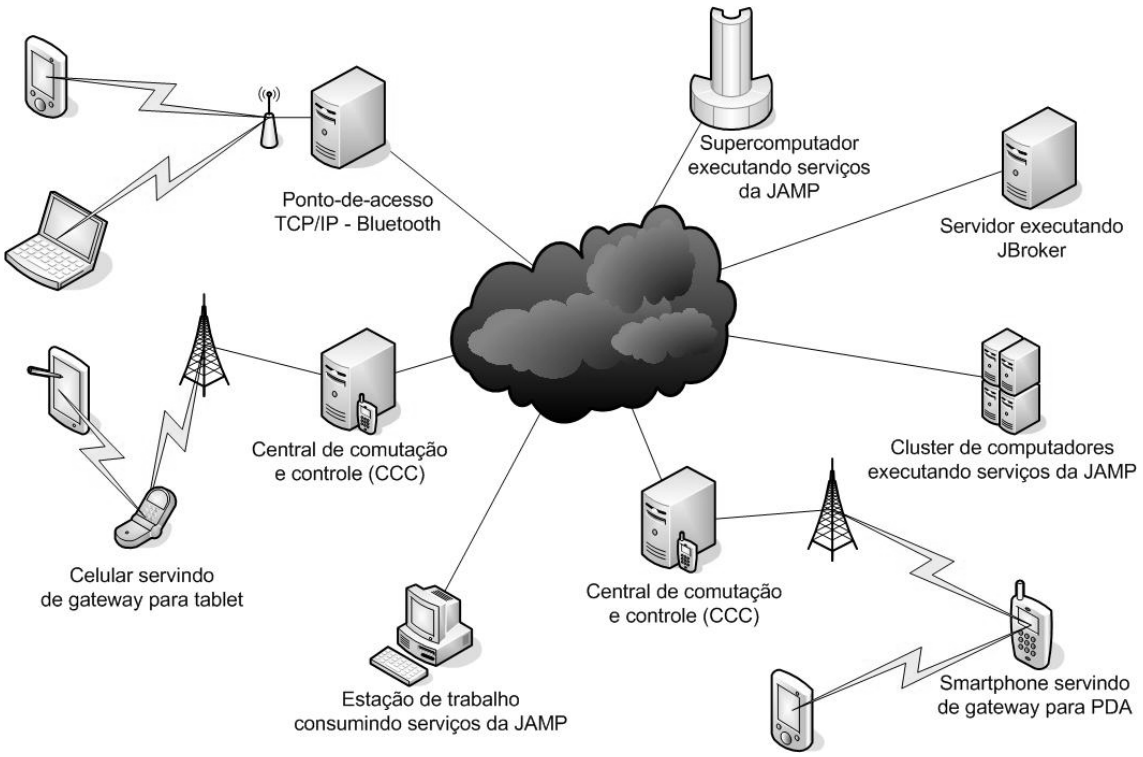

Figura 14: Cenário de operação do *framework* de conectividade BConn.

4.2 Concepção do BConn

A modelagem do *framework* foi feita com base na arquitetura Jini Surrogate [SUR 06]. Ela trata a concentração de funcionalidades em pontos da rede que apresentam maior poder de processamento e que, portanto, servem de *proxies* para dispositivos com limitações. Essa argumentação é válida quando o ponto de acesso é um PC e não quando é um celular ou *smartphone*. Porém, quando o dispositivo móvel com o qual o usuário acessa o serviço é um PDA que não dispõe de interfaces de rede diferentes de Bluetooth, o celular e o *smartphone*

representam um elemento com maior capacidade de conectividade. Neste caso, o ponto de acesso pode ser comparado com um ponto da rede que apresenta maior poder; poder de conectividade e não de processamento.

A arquitetura Jini Surrogate apresenta como requisitos:

a) **independência do tipo de dispositivo**: a arquitetura deve ser capaz de suportar uma variedade de componentes de *software* e *hardware* com diferentes capacidades;
b) **independência do tipo de rede**: a arquitetura deve ser capaz de acomodar diversas tecnologias de rede. A independência do tipo de rede inclui o suporte a diferentes protocolos simultaneamente no mesmo meio físico, como Wi-fi, Bluetooth, etc;
c) **preservação de *plug-and-work***: a arquitetura deve preservar o modelo plugue-e-trabalhe da tecnologia Jini. A arquitetura Jini inclui os conceitos da descoberta, *download* de código e do aluguel de recursos distribuídos. Já a plataforma JAMP trata do uso de objetos remotos, sem o *download* de seus respectivos códigos.

A arquitetura Jini Surrogate tem uma entidade chamada *host-capable machine;* ela é responsável por ligar a parte de interconexão – representada pelo *private protocol* –, como um *proxy* para dispositivos de menor capacidade acessarem serviços Jini. Por definição, o *host-capable machine* tem os recursos computacionais (*host resources*) para a execução de código Java, podendo oferecer os recursos necessários para a execução do mesmo. Ele ainda contém o Surrogate *host*, entidade que contém os recursos do *host* e o Surrogate propriamente dito.

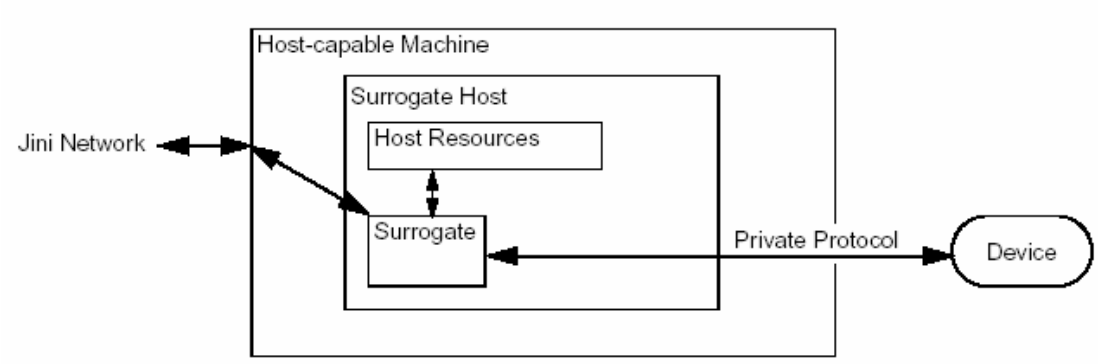

Figura 15: Funcionamento do Surrogate [SUR 06].

O que diverge no BConn é que este não apresenta só um Surrogate, nem mesmo um entidade que representa os recursos do *host*. Ele simplesmente utiliza uma arquitetura cliente-servidor para as conexões entre o ponto de acesso e os dispositivos móveis e outra de *middleware* para utilização de serviços pelos pontos de acesso. Ou seja, o *host* que representa a máquina com capacidades para executar código Java é o ponto de acesso e ele próprio

realiza a busca do serviço requisitado pelo dispositivo móvel – representado, na arquitetura Jini Surrogate, por *device* (Figura 15). Ou seja, o *host-capable machine* e todas as entidades contidas no mesmo, são representados pelo BConn na JAMP numa única entidade de *software*.

Com isso, o BConn também é capaz de oferecer os serviços da JAMP a dispositivos móveis que não disponham de características necessárias para fazer parte da mesma, como aqueles que não têm uma MVJ. Porém, pode fazê-lo devido a todo o processo de obtenção de acesso ao serviço ficar a cargo do ponto de acesso, que executa o BConn. Nesse sentido, aplicações clientes executadas em dispositivos móveis, podem ser desenvolvidas em outras linguagens de programação, como C/C++, que poderão consumir serviços da JAMP como se fossem dispositivos equipados com MVJ.

Partindo da necessidade de se criar um servidor capaz de atender a requisições nas diversas tecnologias de rede sem fio, a classe Frontier abre conexões para as tecnologias de rede sem fio suportadas. Dessa forma, ela é um concentrador de conexões para dispositivos móveis, dispondo de interfaces de rede sem fio, permitindo assim que os dispositivos móveis obtenham acesso aos serviços da JAMP.

Para que serviços sejam localizados por um dado dispositivo móvel que precisa consumi-lo, a classe Finder, uma fábrica de clientes de serviços da JAMP, é instanciada pela Frontier para a realização de consultas ao JBroker. A classe Finder requisita serviços remotos no contexto da JAMP e é um cliente de serviços publicados no JBroker.

Para a concepção do BConn, foram utilizados diagramas *Unified Modeling Language* (UML) de Caso de Uso, de Classe e de Seqüência. Os casos de uso levantados foram dois (Figura 16):

a) Procurar Dispositivos e Serviços;
b) Estabelecer Conexão.

O primeiro trata da busca que um dispositivo móvel faz para obter acesso a determinado serviço. Nesse sentido, o dispositivo móvel procura dispositivos na sua área de alcance e, em seguida, serviços oferecidos pelos mesmos.

No caso de uso de Estabelecer Conexão, uma conexão em modelo cliente-servidor é estabelecida entre o dispositivo móvel e o ponto de acesso. Para essa conexão, o dispositivo móvel conecta-se ao serviço BConn, o qual encaminhará requisições ao objeto remoto que implementa o serviço requisitado pelo cliente móvel.

Finalizada a conexão com o ponto de acesso, é realizada então uma operação de busca daquele serviço por meio de consulta ao JBroker pelo *framework* no ponto de acesso,

procurando uma referência ao serviço desejado em outro nó da JAMP. Ambos os casos de uso são utilizados no acesso de um dispositivo móvel a um serviço implementado em outros nós da JAMP.

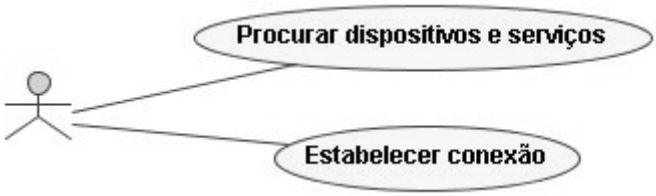

Figura 16: Diagrama de casos de uso do *framework* de conectividade BConn.

A realização dos casos de uso é utilizada na concepção das classes. Estas são a Frontier e a Finder. O relacionamento entre elas pode ser observado na Figura 17.

A classe Frontier é responsável pela interoperação entre redes TCP/IP e Bluetooth. Ela trata do envio e recebimento de dados gerados em comunicação Bluetooth para tráfego em rede TCP/IP, e do estabelecimento de conexão a partir do dispositivo móvel com o ponto de acesso.

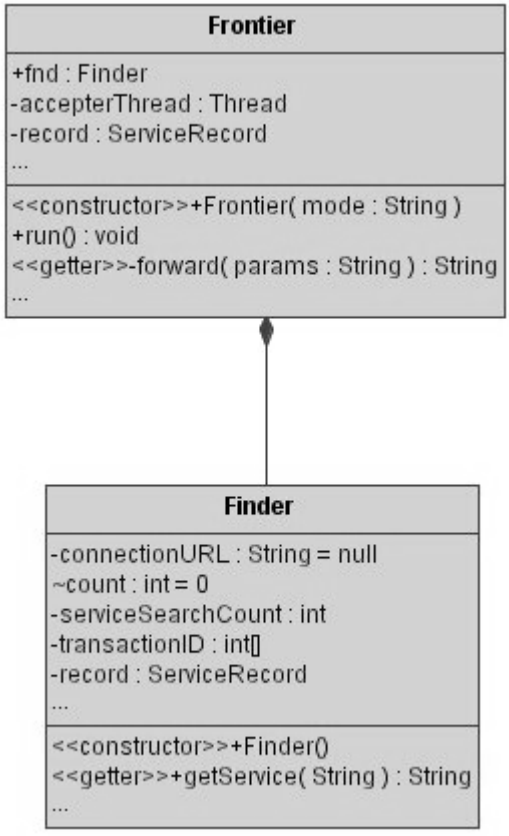

Figura 17: Diagrama de classes do framework BConn.

Os métodos da classe Frontier tratam da realização do caso de uso Estabelecer Conexão Bluetooth. São eles:

a) **run:** *thread* que aguarda o pedido de conexão de um dispositivo móvel para o consumo de serviços oferecidos no ponto de acesso e faz a chamada para o método **forward** (seu conteúdo é exibido na Listagem 5.1 a seguir);
b) **forward**: realiza o encaminhamento de dados do ponto de acesso ao objeto que implementa o serviço consumido pelo dispositivo móvel.

Listagem 1: Método que trata da conexão Bluetooth.

```
  public void run(){
    //Variável de controle que indica se o dispositivo Bluetooth foi
    //obtido e se uma conexão para um serviço encontra-se disponível.
    boolean isBTReady = false;
    StreamConnectionNotifier Server = null;
    try{
      Server =
(StreamConnectionNotifier)Connector.open("btspp://localhost:00012354000
01000800000805F9B34FB");
      record = ld.getRecord(Server);
      DataElement base = new DataElement(DataElement.DATSEQ);
      record.setAttributeValue(SERVICE_ID, base);
      isBTReady = true;
      while(true){
        StreamConnection conn = null;
        System.out.println("Aguardando conexão...");
        conn = Server.acceptAndOpen();
        System.out.println("Conectou!");
        int i;
        String codigo = null;
        boolean done = false;
        BufferedReader is = null;
        //Abrindo stream de entrada de dados
        DataInputStream dip = conn.openDataInputStream();
        //Abrindo stream de saída de dados
        DataOutputStream dop = conn.openDataOutputStream();
        System.out.println("Aguardando os dados..");
        //Código enviado pelo cliente móvel
        codigo = dip.readUTF();
        System.out.println("Código: "+ codigo);
        //Chamada ao método que realiza o encaminhamento de dados
        //de entrada para consumo do serviço
        dop.writeUTF(forward(codigo));
      }
    }catch(IOException e){
      System.err.println("Falha no estabelecimento da conexão.");
      System.err.println("IOException: " + e.getMessage());
      return;

    }
  }
```

A classe Finder trata do processo de busca do objeto remoto que implementa o serviço requisitado pelo dispositivo móvel. Após sua instanciação, é realizada a busca de serviços publicados no JBroker. A partir do retorno do serviço requisitado, o dispositivo móvel o utiliza. Sendo assim, requisições são encaminhadas ao objeto que disponibiliza o serviço por meio de sua referência, utilizada pelo Frontier.

Quando um dado dispositivo busca certo serviço, a classe Finder de seu ponto de acesso faz uma consulta ao JBroker que obterá a referência de serviço localizado em *desktops* ou servidores, disponibilizando o serviço procurado. Com essa referência do serviço, o ponto de acesso faz uso do serviço, oferecido pelo objeto que o implementa, na forma de um *proxy* do dispositivo móvel.

A seqüência de operação do BConn é exibida na Figura 18.

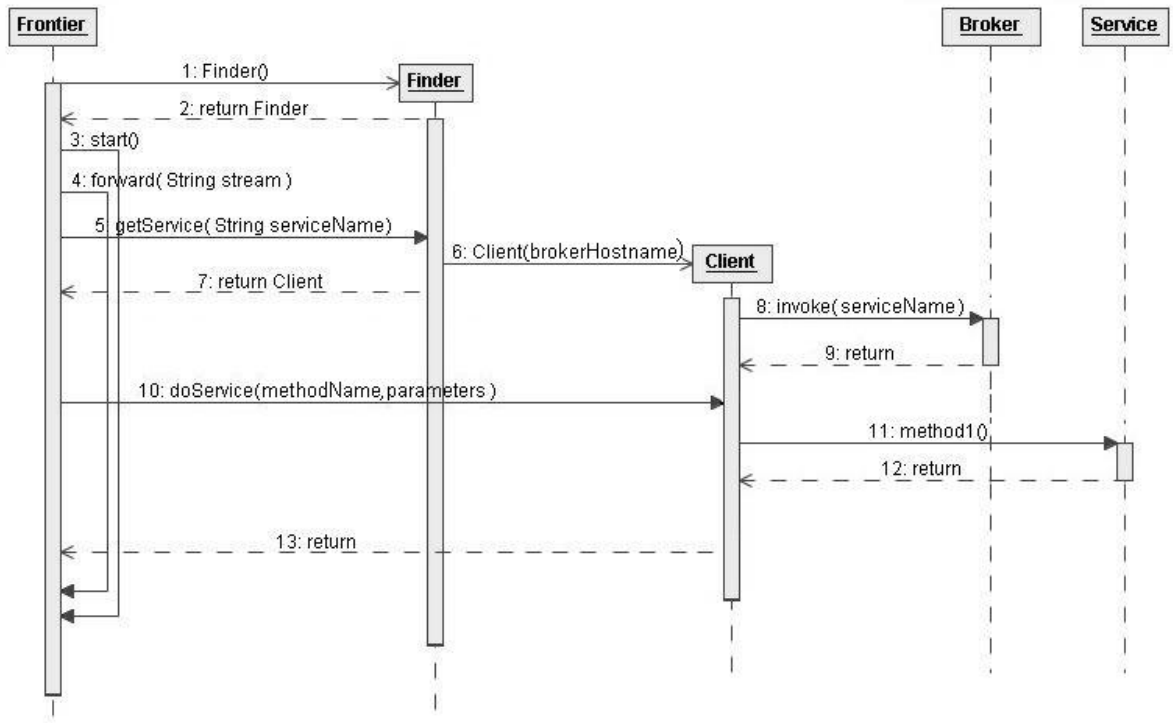

Figura 18: Diagrama de seqüência de operação do Bconn.

4.3 Funcionamento do BConn

O funcionamento do BConn ocorre com base na necessidade de um dispositivo móvel acessar um serviço que faz parte de um Sistema Distribuído em ambiente privativo. Pode ser citado como exemplo o acesso a BDs onde os dados não poderiam estar armazenados no dispositivo móvel, mas que devem ser disponibilizados ao usuário freqüentemente, como é o caso do acesso a dados de prontuário médico.

A seqüência de funcionamento do BConn, para acesso de dispositivos móveis Bluetooth a serviços da JAMP é dada pela Figura 19:

1. Dispositivo móvel, por meio de um modelo cliente-servidor, busca um determinado serviço.

2. Ponto de acesso, servidor no modelo cliente-servidor mencionado no item anterior, busca o serviço na JAMP, assumindo papel de cliente da mesma.

3. É realizada uma consulta ao JBroker.

4. JBroker responde, divulgando a referência ao serviço em um nó da JAMP.

5. A partir da referência fornecida pelo JBroker, o ponto de acesso – papel de *proxy* – requisita o serviço ao nó onde é oferecido.

6. Ponto de acesso obtém resposta do serviço.

7. Ponto de acesso encaminha as funcionalidades do serviço ao dispositivo móvel.

Mais especificamente o passo 2 trata do acesso ao serviço, o qual implica em fornecer algumas informações ao BConn. Este irá analisar o *stream* de dados, *string* enviada pelo dispositivo móvel. Essa *string* conterá o nome de serviço, seguido dos parâmetros de entrada para o serviço, todos divididos por separadores léxicos, como caracteres *pipe* "|". O BConn extrai o nome do serviço que se deseja acessar e o utiliza na consulta ao JBroker.

O passo 7 trata da resposta que é enviada ao cliente no dispositivo móvel. Nele, a partir da resposta do serviço, o conjunto de dados gerado durante o processamento dos parâmetros enviados ao serviço remoto é retornado e repassado pelo BConn ao cliente que o requisitou. É então tratado no cliente, que extrai da *string* retornada os dados, quebrando a *string* por meio dos separadores léxicos que esta contém.

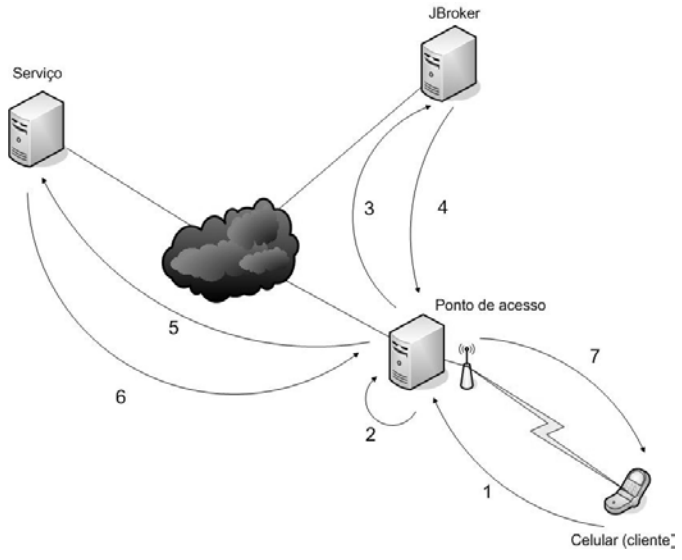

Figura 19: Seqüência de eventos de funcionamento do BConn no acesso a serviços da JAMP.

Para interagir com o *framework*, é necessário o uso de um aplicativo, executado no dispositivo móvel, que se conecta ao serviço remoto por meio do BConn. Uma vez que as MVJ disponíveis para dispositivos da plataforma Palm OS não dão suporte à API Bluetooth na linguagem Java, o cliente para esse tipo de dispositivo móvel, pode utilizar um componente de conectividade baseado no EasyNet[4] [LOP 05], ou deve ser implementado em MIDlet Java executando em dispositivos móveis como celulares ou *smartphones* que implementam a JSR-82.

EasyNet constitui uma camada de abstração que facilita o desenvolvimento de aplicações de conectividade por utilizar o paradigma de Orientação a Objetos. Ele disponibiliza métodos de classes que encapsulam o conjunto de instruções necessárias a uma conexão em tecnologias como Bluetooth, Infravermelho, Wi-Fi e GPRS. Tais instruções eram antes manipuladas pelo desenvolvedor de aplicações uma a uma na API 68K [WIL 04] toda vez que o mesmo precisava implementar o estabelecimento de conexão de rede em aplicações na plataforma Palm OS.

Quando do início do desenvolvimento, tentativas foram feitas com dispositivos Palm. Sobre o SO PalmOS Garnet, tentou-se instalar as máquinas virtuais J9, da IBM, e Sun KVM, da Sun Microsystems, para uso da linguagem Java em todos os segmentos da conexão. Nenhuma delas oferece suporte a API Bluetooth, não atendendo às necessidades de teste com essa tecnologia de rede. Portanto, quando houver dispositivos que executam SO PalmOS de versões onde não seja possível instalar uma MVJ com suporte a Bluetooth, a melhor solução é utilizar o componente EasyNet.

O BConn portanto é um *framework* que auxilia o desenvolvimento de aplicações onde dispositivos que não dispõem de interface de rede TCP/IP podem acessar serviços publicados na JAMP. Isso ocorre devido à implementação de um serviço de *proxy* que é executado nos pontos de acesso à tecnologia de rede em questão; no caso dessa implementação, o Bluetooth.

[4] EasyNet é um componente de conectividade para dispositivos móveis tipo PalmTop desenvolvido no âmbito

5 ESTUDO DE CASO

Com o objetivo de validar o *framework* desenvolvido, foi implementado um estudo de caso que consiste basicamente na recuperação de informação de um paciente, disponível na forma de um serviço da JAMP, a partir de um dispositivo móvel com interface de rede Bluetooth.

5.1 Descrição do cenário de teste

Para validar o sistema, foram utilizados os seguintes dispositivos:

a) 1 Celular modelo Motorola SLVR L7 executando o cliente para acesso ao prontuário médico:
   - Processador Motorola de 52 MHz;
   - 5 MB de memória RAM interna e 128 MB externa;
   - Sistema Operacional Motorola;
   - KVM (*Kilobyte Virtual Machine*) v. 1.1, CLDC 1.1, MIDP 2.0;
   - Interface Bluetooth v. 1.2 classe 2 (alcance de 10 m e potência máxima de 2.5 mW).

b) 1 PC *desktop* executando o BConn:
   - Processador AMD Athlon 64 XP 3500+ de 2.20GHz;
   - 1GB de memória RAM;
   - 1 *Dongle* (interfaceUSB/Bluetooth) Bluetooth v. 1.2 classe 1 (alcance de 100 m e potência maxima 100 mW);
   - Sistema Operacional Windows XP Professional;
   - MVJ v. 1.5.0_07-b03;
   - API Java avetanaBluetooth v. 1.3.10;
   - P2KTools v. 0.8.6-b406, aplicativo usado para destravar a opção de carregamento de MIDlets em celulares de arquitetura P2K, da Motorola;

- MIDway v. 2.8, aplicativo usado para efetuar o carregamento de MIDlets em celulares Motorola, através de conexão USB.

c) 1 PC *desktop* executando o JBroker:
   - Processador AMD Sempron 2600+ de 1.83 GHz;
   - 1GB de memória RAM;
   - Sistema Operacional Windows XP Professional;
   - MVJ v. 1.5.0_07-b03.

d) 1 PC *desktop* executando o serviço Prontuário:
   - Processador AMD Sempron 2600+ de 1.83 GHz;
   - 1GB de memória RAM;
   - Sistema Operacional Windows XP Professional;
   - MVJ v. 1.5.0_07-b03.

e) 1 PC servidor executando servidor Web com Servlet de acesso ao SGBD (Sistema Gerenciador de Banco de Dados):
   - 2 Processadores AMD Opteron 64 1.8GHz;
   - 2GB de memória RAM;
   - Sistema Operacional Linux Debian 3.4.4-0;
   - SGBD MySQL v. 5.0.18;
   - Servidor Web Tomcat v. 2.0.55;
   - MVJ v. 1.5.0_06-b05.

Os dispositivos foram arranjados como mostra a Figura 20. O PC onde é executado o BConn dispõe de uma interface USB/Bluetooth v. 1.2, conhecida como *dongle*, para estabelecer conexão com o celular. Para o acesso ao servidor Web (servidor **`labpalm.dc.ufscar.br`**), existem restrições configuradas em seu *firewall*, impedindo conexões em quaisquer portas que não sejam as portas 80 e 8080, comumente usadas para o acesso via HTTP [FIE 99], por isso o acesso ao serviço é implementado na forma de um Servlet [HUN 01]. Sendo assim, o serviço Prontuário da JAMP faz uso do Servlet para obter e disponibilizar as informações. Caso fosse utilizada uma máquina que contivesse o BD com informações de pacientes sem restrições de acesso, o serviço Prontuário poderia ser executado nesta mesma máquina. Com esse particionamento do serviço Prontuário, a utilização de um Servlet em outra máquina não tira a característica de transparência promovida pelo uso do JBroker, pois o acesso ao serviço Prontuário é obtido por meio de invocação de objeto remoto através de consulta ao JBroker.

A rede de comunicação utilizada foi uma rede local de 100 Mbps.

O cenário criado para os testes é exibido na Figura 20.

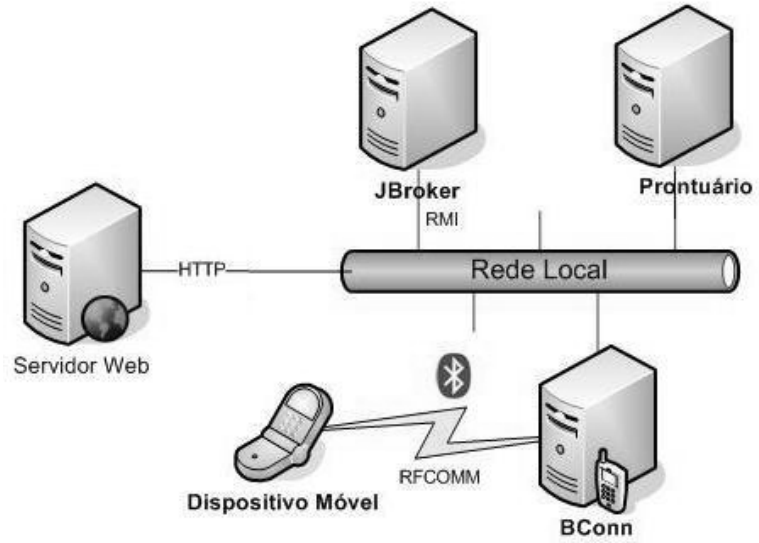

Figura 20: Cenário utilizado no estudo de caso.

5.2 Descrição do sistema

A parte da aplicação presente no dispositivo móvel comunica-se com um ponto de acesso através do protocolo RFCOMM. Essa aplicação utiliza um serviço implementado em um PC, chamado Prontuário, o qual faz parte da JAMP. A aplicação cliente no dispositivo móvel acessa o serviço, requisitando-o ao BConn, que é executado no ponto de acesso, o qual obtém uma instância do objeto de serviço. Para isso, é necessário que o cliente móvel tenha apenas a interface gráfica e conheça as diretivas de comunicação do serviço que precisa acessar para fornecer os dados de entrada necessários e receber o resultado, formatando-o para o usuário. Dessa forma, promove-se a interação do aplicativo do dispositivo móvel (cliente) com o serviço Prontuário oferecido na JAMP (servidor), tendo entre eles o BConn.

A partir do estabelecimento de conexão via Bluetooth entre o cliente móvel e o BConn, este realiza a busca pelo serviço Prontuário no JBroker, o qual retorna uma referência do serviço propriamente dito. A partir dessa referência, o BConn requisita o serviço disponibilizado pelo Prontuário.

O Prontuário consiste de um cliente para consulta de prontuários médicos, disponibilizado no contexto da JAMP como um serviço. Para que fosse possível o acesso a esse serviço, foi utilizado um servidor Web para disponibilizar as informações contidas em um BD. Com isso, requisições são enviadas pelo BConn ao serviço Prontuário. Este realiza a consulta ao BD (Banco de Dados) que se encontra na máquina servidora contendo o servidor Web e o SGBD. Esse serviço é oferecido na forma de um Servlet – classe de negócio

disponibilizada via HTTP – que responde a requisições HTTP originadas no serviço Prontuário. Esse Servlet faz acesso ao SGBD local e retorna o resultado ao Prontuário. A partir desse momento, o Prontuário responde ao BConn que encaminha as informações ao cliente móvel que requisitou o serviço. O cliente móvel então as exibe na tela.

Todo o sistema foi desenvolvido em linguagem de programação Java, fazendo uso das três edições disponíveis na mesma. Ou seja, Edição Micro (J2ME – *Java to Micro Edition*) no aplicativo executado no celular, Edição Padrão (J2SE – *Java to Standard Edition*) no ponto de acesso e Edição Empresarial (J2EE – *Java to Enterprise Edition*) no ponto de acesso, na máquina que disponibiliza na JAMP o serviço Prontuário e no *site* onde é executado o servidor Web e o SGBD.

5.3 Análise dos resultados do estudo de caso

Para a realização do experimento foi montado um cenário onde um celular executando uma aplicação cliente, na forma de um MIDlet, envia solicitações para o ponto de acesso, que contém o BConn, dentro de sua área de alcance. A comunicação entre estes nós do sistema é estabelecida através da tecnologia de rede Bluetooth em modelo cliente-servidor. Os dispositivos foram dispostos a uma distância de aproximadamente 50 cm.

A partir do PC que serve como ponto de acesso há outra máquina interligada a este, a qual disponibiliza o serviço Prontuário. Ela se comunica com o BConn, por meio de RMI. Essa comunicação, portanto, ocorre em rede TCP/IP e tem como característica o uso das funcionalidades implementadas no objeto de serviço (remoto) como se este estivesse presente na mesma MVJ onde é executado o BConn, o ponto de acesso. Tal utilização ocorre somente após a consulta ao JBroker para obtenção da referência do objeto remoto Prontuário. O serviço Prontuário faz então acesso ao SGBD localizado no servidor LabPalm (**`labpalm.dc.ufscar.br`**) por meio de um servidor Web, no qual encontra-se um Servlet que acessa o SGBD contendo os dados dos pacientes.

5.3.1 Facilidade de uso e funcionalidade do *framework*

Com relação à facilidade de uso do *framework* no desenvolvimento de novas aplicações, essa se mostrou relativamente simples devido à implementação tanto do cliente quanto do serviço Prontuário oferecido na JAMP serem praticamente as mesmas que as criadas para o acesso a partir de um PC. A mudança empregada no cliente é a interface com o usuário e o tratamento dos dados retornados pelo serviço. Em dispositivos móveis, como celulares, *smartphones* e PDAs, a implementação Java se dá na forma de um MIDlet, aplicação que se limita às características da MVJ instalada no dispositivo e às dele próprio.

Os dados retornados ao cliente precisam ser processados pelo mesmo para que haja a separação dos itens retornados, a partir do *stream*, para posterior exibição ao usuário.

Devido ao processamento realizado pelo BConn, sua parte cliente na JAMP, a qual invoca o objeto remoto que implementa o serviço Prontuário, encaminha os dados de entrada da mesma forma que quando era feita diretamente por um cliente em rede TCP/IP, usando diretamente o serviço; havendo ainda a diferença que implica na consulta aos dados por meio de um Servlet em uma terceira máquina, a qual contém o SGBD.

Um *stream* contendo os parâmetros devidamente seqüenciados para interpretação no destino é enviado pelo BConn ao serviço Prontuário. Como o processamento dos dados para geração dos parâmetros corretos é feito pelo BConn, o serviço é o mesmo que atende a um cliente implementado para execução em PC.

Dessa forma, a facilidade de uso do *framework* para desenvolver aplicações viabiliza o acesso de dispositivos móveis, bem como era feito antes por clientes em PCs, adicionando apenas um servidor para conexões Bluetooth e a transformação dos dados de entrada do cliente do serviço na JAMP.

Nesse sentido, a funcionalidade do *framework* consiste em encaminhar dados de uma conexão cliente-servidor Bluetooth para o consumo de serviços da JAMP, usando objetos distribuídos via RMI.

Diante da praticidade para o desenvolvimento do acesso por outras tecnologias de rede sem fio, o *framework* mostra-se bastante flexível. Com ele, novas tecnologias podem ser agregadas por meio da implementação para conexão nessa nova tecnologia, criando-se métodos que tratem dos aspectos envolvidos no estabelecimento da conexão para a tecnologia em questão.

Sendo assim, a plugabilidade do sistema está assegurada, pois criando o método de conexão com o BConn na tecnologia de comunicação sem fio, este realiza consultas ao JBroker para que o dispositivo móvel utilize o serviço localizado na JAMP, conferindo a usuários móveis, transparência no acesso ao serviço remoto.

5.3.2 Desempenho

Devido ao uso da rede, uma sobrecarga é adicionada ao BConn, comparado a uma aplicação *desktop*. Portanto, dependendo das condições da mesma, ou seja, do tráfego nos segmentos de rede entre o BConn e o serviço consumido, bem como da sobrecarga para a consulta ao JBroker, haverá variações no tempo de resposta.

Além disso, a busca por dispositivos e serviços na tecnologia Bluetooth apresenta sobrecarga considerável, a qual pode ser diminuída através da obtenção de referência relativa a dispositivos móveis pré-reconhecidos ou de dispositivos acessados anteriormente, armazenados no *cache* do dispositivo móvel que acessa o BConn. Com isso, o tempo de descobrimento é reduzido apenas a basicamente o descobrimento de serviço.

No caso estudado, o tempo para a conexão Bluetooth, que consiste em localizar dispositivos na área de alcance, levou aproximadamente 10320 ms e a busca de serviço correspondeu a aproximadamente 1186 ms, como pode ser visto na Tabela 5.

Tabela 5: Medidas de tempo para descobrimento de dispositivo e de serviço.

Número de vezes	Tempo para descobrimento de dispositivo	Tempo para descobrimento de serviço	Tempo total
1	10320	1190	11510
2	10290	1185	11475
3	10180	1188	11368
4	10320	1186	11506
5	10350	1186	11536
6	10320	1186	11506
7	10320	1185	11505
8	10310	1186	11496
9	10320	1186	11506
10	10320	1187	11507

A partir dessa localização (obtida com o descobrimento de dispositivo, seguido do descobrimento de serviço), há a execução do método **run**, exibido na Listagem 1. O mesmo constitui:

a) processamento da requisição para conexão Bluetooth, em modo servidor, aguardando requisições do celular;
b) consulta ao JBroker e manipulação do objeto remoto;
c) encaminhamento dos dados obtidos do celular para processamento no objeto de serviço (constituindo também acesso ao SGBD) e subseqüente retorno dos dados ao BConn;
d) além da resposta ao celular.

Esse tempo corresponde a aproximadamente 520 ms, dos quais a invocação do objeto remoto variou em torno de 300 ms, apresentando extremos de 296 a 1546 ms. A execução do método **forward**, que corresponde a manipular o objeto remoto que abre conexão com o Servlet que implementa a consulta ao SGBD, seguido do envio dos dados ao mesmo e obtenção da resposta dele, oscilou em torno de 16 ms.

Para a realização dos testes, a metodologia empregada foi a repetição dos experimentos e o apontamento dos tempos, onde os valores repetidos mais vezes são os apresentados acima. Tal metodologia foi suficiente para concluir que o *framework* é útil, pois não exige do usuário grandes intervalos para a obtenção de dados do serviço remoto, com exceção do tempo de localização de dispositivos, o qual pode ser minimizado com o *cache* de dispositivos pré-acessados.

Diante dos tempos levantados nos testes, o *framework* mostrou-se com grande aplicabilidade, pois o mesmo permite a obtenção de informações de serviços remotos suportados por RMI mesmo a partir de tecnologia de rede diferente daquela necessária para o tráfego RMI, o TCP/IP.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou a proposta de um *framework* de conectividade para dispositivos móveis a serviços de rede, com descoberta e acesso a tais serviços através da Plataforma JAMP. Um estudo de caso foi desenvolvido utilizando-se o *framework* BConn para dispositivos com tecnologia Bluetooth. A partir dos resultados obtidos, é possível concluir-se que a viabilidade do acesso à JAMP por dispositivos móveis é aplicável, podendo ser estendida a outras tecnologias além do Bluetooth. A seção 6.1 a seguir apresenta algumas conclusões sobre os resultados obtidos com este *framework*.

6.1 Conclusões

A partir dos resultados obtidos neste trabalho, usuários móveis podem utilizar uma infra-estrutura de comunicação sem fio para acessar serviços remotos tendo como vantagem a estrutura de distribuição de dados provida pela Internet, além de ter acesso aos serviços oferecidos na JAMP com mobilidade. JAMP já disponibiliza meios de distribuição de serviços para trabalho colaborativo e multimídia; oferece um arcabouço para a implementação de aplicações voltadas à utilização de usuários geograficamente distribuídos, servindo este trabalho como extensão desses serviços em comunidades *ad hoc*. O acesso de dispositivos móveis a quaisquer serviços de *middleware* implementados na JAMP é oferecido pelo *framework* BConn.

Ainda há muito que se desenvolver no que tange ao reconhecimento de serviços por clientes móveis em redes sem fio [HEL 02, MAS 02]. Com isso, a proposta colabora com o panorama de reconhecimento e utilização de serviços remotos disponíveis em rede.

A utilização da plataforma JAMP colabora para tal desenvolvimento devido a esta prover arquitetura voltada ao desenvolvimento de aplicações distribuídas. Com isso, há uma base sólida para o desenvolvimento de diferentes *frameworks* sobre a mesma.

Diante da estrutura provida por JAMP e de equipamentos disponíveis para a realização de testes, o trabalho gerou resultados satisfatórios para validação do *framework* proposto. A viabilidade de se acessar um serviço a partir de um dispositivo móvel é

assegurada com o *framework* de serviço BConn que é executado no ponto de acesso. Com ele, serviços requisitados por dispositivos móveis são encaminhados para o servidor que o implementa e a resposta de suas funcionalidades, retornadas ao cliente que as consome.

6.2 Trabalhos Futuros

Como trabalho futuro aponta-se para a melhoria da interação entre o usuário e a arquitetura, devido ao aumento de praticidade de utilização da mesma para o consumo de qualquer serviço em qualquer lugar, desde que o cliente se encontre dentro da área de alcance de um ponto de acesso.

O uso de uma única aplicação instalada no dispositivo móvel, com a qual o usuário realizaria consultas sobre quaisquer serviços publicados no JBroker é algo que agregaria maior utilidade ao *framework*. Essa estrutura permitiria, a partir dos resultados da consulta, o *download* de pacotes de aplicativos Java a partir do BConn para o dispositivo móvel. Com isso, existindo apenas um aplicativo cliente que realize consultas, encaminhadas pelo ponto de acesso, a serviços publicados no JBroker, seria suficiente para o dispositivo móvel fazer parte dessa arquitetura distribuída.

Dessa forma, o usuário poderia escolher qual serviço desejaria utilizar e a partir desse momento o dispositivo móvel realizaria o *download* das classes, necessárias à utilização do serviço, – classes de *view* e *control* do modelo MVC (*Model View Controller*) [KRA 88] – armazenadas no ponto de acesso (no contexto do BConn). Esse aplicativo cliente é aquele que acessa o serviço disponibilizado em algum PC participante da JAMP, instalado no dispositivo móvel. A partir daí, o dispositivo móvel comunicar-se-ia com o ponto de acesso para requisitar o serviço ao JBroker.

Essa funcionalidade proveria um aumento significativo de ubiqüidade à JAMP, no sentido de que não seria mais necessário instalar aplicativos que fazem acesso a cada serviço publicado no JBroker, pois a própria arquitetura ofereceria essa facilidade ao usuário. O usuário então instalaria aquilo que é necessário para acessar o serviço que deseja utilizar naquele momento.

Outra tentativa para a flexibilização do ponto de acesso seria o uso de reflexão. O uso da API Reflection, de Java, permite que o ponto de acesso, mediante requisição de um aplicativo cliente, seja capaz de identificar como são as classes de serviço. Com isso, seria possível oferecer o serviço sem a necessidade de um cliente da JAMP no ponto de acesso para comunicar com o objeto de serviço.

O uso de reflexão torna capaz a obtenção dos métodos, atributos, construtores e superclasses de uma classe; encontrar instâncias da classe, invocar métodos, atribuir e obter valores de atributos, mesmo que seus nomes sejam desconhecidos em tempo de compilação [REFLECTION].

A utilização da API Reflection tornaria o ponto de acesso apto a identificar as interfaces de comunicação do objeto de serviço, permitindo que qualquer cliente no dispositivo móvel acessasse um serviço distribuído remoto implementado no objeto de serviço. Isso porque o ponto de acesso descobriria quais os métodos e atributos do objeto de serviço e intermediaria a comunicação, realizando os ajustes necessários para o estabelecimento da comunicação do sistema. Dessa forma, o ponto de acesso seria auto-contido e não necessitaria de cada cliente JAMP para cada serviço que fosse acessar.

Uma outra maneira de melhorar a interação entre o usuário e a arquitetura, seria por meio da instalação de serviços sob demanda. Dessa forma, uma única aplicação seria instalada no dispositivo móvel, seria feito o *download* de aplicação MIDlet (view layer) para o dispositivo móvel, *download* de cliente JAMP para o ponto de acesso (esse podendo ser suprimido com o uso de reflexão) e não seria necessário instalar manualmente cada MDIlet para acessar cada serviço.

A JSR 232 define um *framework* de gerenciamento de componentes em dispositivos baseados em J2ME para adaptar suas capacidades com instalação de novos componentes sob demanda. A JSR é voltada para uso em dispositivos do tipo CDC, que apresentam maior capacidade de processamento, memória e comunicação.

Baseado na proposta da JSR 232, a idéia de utilizar o mesmo conceito no BConn permitiria uma utilização mais automatizada da arquitetura nos dispositivos móveis, pois dispensaria sucessivas instalações, livrando o usuário de preocupações desnecessárias.

7 REFERÊNCIAS

[ABO 99] ABOWD, G. D., **Classroom 2000**: An experiment with the instrumentation of a living educational environment. IBM Systems Journal, 1999, v.38.

[ALM 02] ALEMIDA, E. S.; PRADO; A. F., **Desenvolvimento de software baseado em componenetes distribuídos**, Dissertação de mestrado, UFSCar, 2003.

[ANYPOINT] **AnyPoint**, http://www.intel.com/anypoint/, acessado em março de 2005.

[ARA 03] ARAUJO, R. B., **Computação Ubíqua: Princípios, tecnologias e desafios**. Minicurso SBRC2003, 2003.

[ARM] http://tisu.it.jyu.fi/embedded/TIE345/luentokalvot/Embedded_3_ARM.pdf, **ARM Architecture**, acessado em junho de 2005.

[ATINAV] **Atinav and Bluetooth**, disponível em: http://www.atinav.com/bluetooth/bluetoothjava.pdf, acessado em maio de 2005.

[AUTHORING] **Authoring on the Fly**, disponível em: http://ad.informatik.uni-freiburg.de/mmgroup.aof/, acessado em maio de 2005.

[AUS 00] AUS-SYSTEM, **Bluetooth Whitepaper 1.1**, Au-system, p. 25, 2000.

[AVETANA] **Avetana JSR-82 implementation**, disponível em: http://www.avetana-gmbh.de/avetana-gmbh/jsr82.xml, acessado em setembro de 2005.

[AVETANABLUETOOTH] **AvetanaBluetooth JSR-82 implementation** , http://sourceforge.net/projects/avetanabt/, acessado em maio de 2005.

[BAC 03] BACON, J.; HARRIS, T., **Operating Systems: Concurrent and Distributed Software Design**. Boston, MA, EUA, Addison-Wesley, 2003, p. 720.

[BAN 02] BANAVAR, G.; BERNSTEIN, A., **Software Infrastructure and Design Challenges for Ubiquitous Computing Applications**, In **Communications of the ACM**, 2002, Vol. 45, nº 12.

[BAS 97] BASS, L. et. al, **The design of a wearable computer**, **Conference on Human Factors in Computing Systems**, Proceedings of the SIGCHI conference on Human factors in computing systems, 1997, p. 139-146.

[BAT 04] BAPTISTA, B. A. D., **Projeto do Subsistema de Comunicação e Distribuição e da Camada de Serviços da Arquitetura OpenReality para suporte à criação de Aplicações de Visualização Distribuída**. Dissertação de mestrado, UFSCar, 2004.

[BEL 04] BELLAVISTA; P. STEFANELLI; C. TORTONESI, M. **The ubiQoS middleware for audio streaming to Bluetooth devices**. Proceedings of the

First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04), IEEE, 2004, p. 138-145.

[BET 00] BETTSTER, C.; RENNER, C., **A comparision of aservice discovery protocols and implementation of the service location protocol**, Summerschool, University of Twente, 2000.

[BHA 01] BHAGWAT P., **Bluetooth**: Technology for Short-Range Wireless Applications, IEEE Internet Computing, 2001, v. 5, n. 3, p. 96-10.

[BLA 01] BLACK, K. et. al, Wireless Access and Terminal Mobility in CORBA, White Paper, 2001.

[BLUETOOTH] **Bluetooth Articles**, http://www.palowireless.com/bluearticles/, acessado em abril de 2005.

[BUR 02] BURKHARD, T. J. et. al, apud de ARAÚJO, R. B., **Pervasive Computing: technology and architecture of mobile internet applications**, Ed. Addison-Wesley, 2002, p. 410.

[CAM 99] CAMPBELL, A. et al., **Cellular IP,** disponível em: http://www.comet.columbia.edu/cellularip/pub/draft-ietf-mobileip-cellularip-00.txt, 1999.

[CHA 04] CHAKRABARTI, S., LIYUN, W. S. V., LEUNG, V.C.M. **A remotely controlled Bluetooth enabled environment**. **Consumer Communications and Networking Conference** (CCNC 2004), 2004, p. 77-81.

[CERF 74] CERF, V. G., KAHN, R, E., **A protocol for packet network intercommunication**. **IEEE Transactions on Communications**, IEEE Press, Los Alamitos, CA, EUA, v. 22, n. 5, p. 637-648, 1974, disponível: ieeexplore.ieee.org/ie15/8159/23818/01092259.pdf,. acessado em abril de 2005.

[CHE 02] CHEN , C. W. et. al, **Support and optimization of Java RMI over Bluetooth environments**, **Concurrency and Computation**: Practice and Experience, Wiley (Special Issue for Java Grande-ISCOPE 2002), 2002.

[CLA 05] CLAUDINO, R. A. T., SOUZA, W. L. DE, PRADO, A. F. DO. **Um Framework Baseado em Componentes para o Dominio de Adaptação de Conteudo na Internet**. **Simpósio Brasileiro de Engenharia de Software**, 2005.

[COMPUTING] **Computing at the Edge**, **Sun Microsystems White Paper**, 2003.

[CONNECTEDa] **Connected Device Configuration (CDC); JSR 36, JSR 218**, disponível em: http://java.sun.com/products/cdc/, acessado em maio de 2005.

[CONNECTEDb] **Connected Limited Device Configuration (CLDC); JSR 30, JSR 139**, http://java.sun.com/products/cldc/, acessado em maio de 2005.

[CORBA] **CORBA 3 Release Information,** 2004, disponível em: http://www.omg.org/technology/corba/corba3releaseinfo.htm, **CORBA 3 Released**, acessado em fevereiro de 2005.

[CORE] MUCHOW, J. W., **Core J2ME technology & MIDP**, Prentice Hall PTR Upper Saddle River, NJ, 2002.

[COS 02] COSTA, F., KON, F., **Novas Tecnologias de Middleware: Rumo à Flexibilização e ao Dinamismo**. **Minicurso,** 20º Simpósio Brasileiro de Redes de Computadores, 2002, pp. 1-61.

[COU 00] COULOURIS, G., DOLLIMORE, J., KINDBERG, T., **Distributed Systems: concepts and design**, 3rd. ed. Boston, MA, EUA, Addison-Wesley Longman Publishing Co., In., 2000.

[DEU 94] DEUTSCH, J. M., **The evolution of customer middleware requirements**, In: proceedings of the **Third International Conference on Parallel and Distributed Information Systems**, Austin, TX, EUA: IEEE Press, 1994, p. 262-263, http://dx.doi.org/10.1109/PDIS.1994.331703. Acessado em janeiro de 2005.

[DIX 02] DIXIT, S., **Wireless IP and its challenges for the heterogeneous environment**, **International Journal for Wireless Personal Communicactions,** 2002.

[DIX 03] DIXIT, S. E PRASAD, R., **Seamless Mobility and IP** Wireless IP and Building the Mobile Internet, Artech, 2003.

[EASY] **Easy Living**, http://research.microsoft.com/easyliving/, acessado em junho de 2005.

[EKL 02] EKLUND, C., et al., **IEEE Standard 802.16**: A Technical Overview of the WirelessMAN Air Interface For Broadband Wireless Access. **IEEE Communications Magazine**, 2002.

[EMM 00] EMMERICH, W., **Software engineering and middleware: a roadmap**. In: ICSE '00: Proceedings of the **Conference on the future of software engineering**. Limerick, Irlanda: ACM Press, 2000, p. 17-129, http://dx.doi.org/10.1145/336512.336542, acessado em junho de 2005.

[ENR 04] ORTIZ, E., **Using the Java APIs for Bluetooth Wireless Technology, Part 1 - API Overview**, http://developers.sun.com/techtopics/mobility/apis/articles/bluetoothintro/, , acessado em 2005.

[FAR 98] FARLEY, J., **Java Distributed Computing**. O'Reilly, First Edition, 1998.

[FER03] FERLINE, O., **Interconexão de Redes Bluetooth** – Uma Aplicação em Telemática de Serviços de Distribuição de Energia, disponível em: www.ppgia.pucpr.br/ensino/defesas/Odair_Ferline.pdf, 2003. Acessado em maio de 2005.

[FER 98] FERREIRA, M.M., TREVELIN, L.C., **The Java Broker System**: Concepts & Java Programming Guide. Relatório Técnico, Universidade Federal de São Carlos, São Paulo, 1998.

[FER 98] FERREIRA, M.M., TREVELIN, L. C., **A Platform for Developing Distributed Multimedia Application**. Techinical Report.DC,UFSCar, 1998.

[FIE 99] FIELDING, R., et. al, **RFC 2616: Hypertext Transfer Protocol - HTTP/1.1**, 1999.

[FIN 95] FINKELSTEIN, R., **The new middleware**, SIGMOD Rec., **ACM Press**, New York, NY, EUA, 1995, v. 24, n. 4, p. 102-106, ISSN 0163-5808, http://dx.doi.org/10.1145/219713.219777. Acessado em maio de 2005.

[FIT 06] FITTON, D., **Java Bluetooth HOWTO**, http://www.caside.lancs.ac.uk/java_bt.php, acessado em dezembro de 2005.

[FMJ 04] JARDIM, F. M., **Framework para implementação de serviços transmissores de vídeos voltados a PCs e dispositivos móveis, perceptivo à mudança contextual de localização**, Dissertação de Mestrado, 2004.

[FRES 01] FRESCO, N., **The CVM Virtual Machine: A Java Techonoly-based VM for Consumer Eletronics**, Sun Microsystems, WJDC, 2001.

[GAM 95] GAMMA, E. et. al, **Design Patterns**: Elements of Reusable Object-Oriented Software, 1st. ed. Boston, MA, EUA, Addison-Wesley Longman Publishing Co., In., 1995.

[GAR 02] GARLAN, D. et al., **Project Aura**: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing, 2002, p. 22-31.

[GER 02] GERLA, M. et al. **UMTS–TDD**: A Solution for Internetworking Bluetooth Piconets in Indoor Environments. Procedings of **IEEE Symposium on Computers and Communications**, ISCC'02, Taormina, Italy, 2002.

[GUTa 99] GUTTMAN, E., **Service location protocol**: automatic discovery of IP networkservices, **Internet Computing**, IEEE, 1999.

[GUTb 99] GUTTMAN, E., et al., **RFC 2608**: Service Location Protocol, Version 2, 1999.

[HAN 01] HANSMANN, U. et al., apud de ARAUJO, R. B., **Pervasive computing handbook**, 2001.

[HEL 00] LEE, C., HELAL, A., **Protocols for Service Discovery in Dynamic and Mobile Networks**, Accepted for publication in the **International Journal of Computer Research**, Special issue on Wireless Systems and Mobile Computing, 2000, p. 18.

[HEL 01] HELD, G., **Data Over Wireless Networks**: Bluetooth,WAP, & Wireless LANs. McGraw-Hill, 2001, p. 344.

[HEL 02] HELAL, S., **Standards for service discovery and delivery**, Pervasive Computing, 2002, p. 95-100.

[HEL 03a] LEE, C., HELAL, A., **A Multi-tier Ubiquitous Service Discovery Protocol for Mobile Clients**, Submitted to the First ACM **International Conference on Mobile Systems and Applications** (Mobisys), 2002.

[HEL 03b] LEE, C., HELAL, A., **Context Attributes**: An Approach to Enable Context-awareness for Service Discovery, Proceedings of the **Third IEEE/IPSJ Symposium on Applications and the Internet**, Orlando, Florida, 2003.

[HOP 04] HOPKINS, B., **Getting Started with Java and Bluetooth**, http://today.java.net/pub/a/today/2004/07/27/bluetooth.html, 2004, acessado em maio de 2005.

[HUN 01] HUNTER, J., **Java Servlet Programming**, 2nd Edition, Ed. O'Reilly, 2001.

[INTEL] http://www.intel.com/products/centrino/, **Intel® Centrino® Mobile Technology**, acessado em maio de 2005.

[ISO] International Organization for Standardization, **ISO 8879**: Information processing: Text and Office Systems - Standard Generalized Markup Language (SGML), **International Organization for Standardization**, 1986.

[OPEN] International Organization for Standardization, **Open Distributed Processing Reference Model**: Overview, Genebra, Suíça, ISO Published Standards, 10746-1, disponível em: http://www.iso.org, 1998.

[J2EE] **The J2EE 1.4 Tutorial**, ARMSTRONG, E. et al., Sun Microsystems, 2003.

[JINIa] **Jini Network Technology**, http://www.sun.com/software/jini/, acessado em maio de 2005.

[JINIb] **Jini**, http://www.ime.uerj.br/~alexszt/cursos/topesp_inter/trabs/992/g14/ , acessado em maio de 2005.

[JSR-82] **JSR 82: Java APIs for Bluetooth**, http://jcp.org/en/jsr/detail?id=82, acessado em abril de 2005.

[JSR-197] **JSR 197: Generic Connection Framework Optional Package for the J2SE™ Platform**, www.jcp.org/en/jsr/detail?id=197, acessado em maio de 2005.

[JUR 04] JURIC, M. B., et al., **RMI Tunneling and Web Services Comparison and Performance Analysis**. **ACM SIGPLAN**, 2004, vol.39(5).

[JUN 01] JÚNIOR, S. T. D., **Análise do Tráfego de Dados em Redes Bluetooth**, www.cin.ufpe.br/~tg/2001-1/STDJ01.doc, 2001.

[KAL 01] KALAFONOS, D., **Internal Nokia Memo**, 2001.

[KAP 03] KÄPPELI, D. **JXTA over Bluetooth**. Diploma Thesis, Swiss Federal Institute of Technology Zurich, 2003.

[KEM 06] KEMPF, J., et al., **Problem Statement for Network-based Localized Mobility Management**, **IETF**, 2006, disponível em: http://www.ietf.org/internet-drafts/draft-ietf-netlmm-nohost-ps-05.txt.

[KIN 92] KING, S., S., **Middleware**: making the network safe for application software, **Data Communications**, 1992, v. 21, n. 4, p. 58-67.

[KRA 88] KRASNER, G. E., POPE, S. T., A **cookbook for using the model–view–controller user interface paradigm in smalltalk-80**. In **Journal of Object Oriented Programming**, p. 26– 49, 1988.

[KRA 03] KRAKOWIAK, S., **What is middleware**. ObjectWeb: Open Source Middleware, http://middleware.objectweb.org, 2003, acessado em maio de 2005.

[LAMONT] LAMONT, A., **Exploring J2ME series**, builder.com.com/5100-6370-1051771.html, 2003, acessado em maio de 2005.

[LOP 05] LOPES, R. R. F., et al.. C. EasyNet: **Um Componente de Conectividade para a Plataforma Palm**, *WebMedia*, 2005.

[LOP 06] LOPES, R. R. F., et al. **Um Componente para o Desenvolvimento de Aplicações TCP/IP em Dispositivos Móveis**. Relatório Técnico nº 1, Projeto LabPalm, Departamento de Computação, Universidade Federal de São Carlos, 2006.

[LYY 02a] LYYTINEN, K. E YOO, Y., **Issues and Challenges in Ubiquitous Computing** In **Communications of the ACM**, Vol. 45, No. 12, 2002.

[LYY 02b] LYYTINEN, K. E YOO, Y., **The next wave of nomadic computing**, Information Systems Research, 2002.

[MOBILE] **Mobile Ad-hoc Networks**, disponível em: http://www.ietf.org/html.charters/manet-charter.html.

[MAS 02] MASCOLO, C., CAPRA, L., EMMERICH, W., 2002, **Mobile computing middleware**. In: Gregori, E., Anastasi, G., Basagni, S., **Advanced Lectures on Networking**: Networking 2002 Tutorials. Pisa, Itália: Springer-Verlag, (Lecture Notes in Computer Science, 2497), 2002, p. 20-58, http://www.springerlink.com/link.asp?id= kqclj3me68yppmc. Acessado em abril de 2005.

[MAT 03] MATTERN, F., STURM, P., **From Distributed Systems to Ubiquitous Computing -** The State of the Art, Trends, and Prospects of Future Networked Systems, Proc. **KIVS 2003**, Springer-Verlag, 2003, p. 3-25; www.vs.inf.ethz.ch/publ/papers/DisSysUbiComp.pdf. Acesado em maio de 2005.

[MED 99] MENDONCA, P. S. et al., **Development of object oriented distributed systems (DOODS) using *framework*s of the JAMP platform**. – ICSE´99, Los Angeles, USA, 1999.

[GETTING] **Getting Started With MIDP**, http://developers.sun.com/techtopics/mobility/learn/midp/getstart/, acessado em junho de 2005.

[MIL 01] MILLER, M., **Discovering Bluetooth,** Ed. Sybex, 2001.

[MMCPLUS] **MMCplus, MMCmobile and MMCmicro Cards**, http://www.mmca.org/, acessado em maio de 2005.

[MOZ 98] MOZER, M. C., **The neural network house: An environment that adapts to its inhabitants**. In **American Association for Artificial Intelligence Spring Symposium on Intelligent Environments**, 1998, p. 110-114.

[MU 05] MU-JUNG C., et al., **The development of smart-phone-based home care evaluation support system**, Proceedings of 7th **International Workshop on Enterprise networking and Computing in Healthcare Industry**, HEALTHCOM 2005, 2005, p. 267-268.

[MUK 99] MUKHOPADHYAY, S., SMITH, B., **Passive Capture and Structuring of Lectures**, Proceedings of **ACM Multimedia**, 1999.

[MYE 02] MYERSON, J. M., 2002, **The Complete Book of Middleware**. New York, NY, EUA: Auerbach Publications, 2002, p. 304.

[NOR 00] NORDSTEDT, D. R., 2000, **The Wireless Jini Projection Service**, University of Florida, Trabalho de graduação.

[NUOFFICE] **NuOffice V2.0**, http://www-06.ibm.com/jp/pspjinfo/salutation/nuoffice.html, 2001, acessado em maio de 2005.

[OBEX4J] SHEN, J., **OBEX4J**, https://obex4j.dev.java.net/, acessado em maio de 2005.

[OPENSLP] **OpenSLP**, http://www.openslp.org/, acessado em abril de 2005.

[OSGI] **OSGi**, www.osgi.org/osgi_technology/?section=2, acessado em maio de 2005.

[OXYGEN] **MIT Project Oxygen:** Pervasive, Human-centred Computing, http://oxygen.lcs.mit.edu, acessado em maio de 2005.

[EVENT] Palm Inc., **Event Loop**, **Palm OS Programmer's Companion**, 2000.

[PAS 01] PASCOE, R., **Building networks on the fly, IEEE Spectrum**, 2001, p. 61-65.

[PAU 05] PAULA, L., B. DE, **Medidas para identificação de sensores invasores em redes de sensores sem fio**, Dissertação de Mestrado, 2005.

[PEI 02] PEIKARI, C., FOGIE, S., **Maximum Wireless Security**, Ed. Sams, 2002, p. 40.

[PER 02] PERKINS, C., **IP Mobility Support for IPv4**, IETF RFC 3220, 2002.

[PRA 02] PRASAD, R. E RUGGIERI, M., **Special Issue on Unpredictable Future of Wireless Communications**, International **Journal for Wireless Personal Communications,** 2002.

[PROCESSORS] **Processors**, http://support.intel.com/support/processors/mobile/pentiumiii/sb/CS-007509.htm, acessado em junho de 2005.

[RHO 99] RHODES, N., MCKEEHAN, J., **Development Environments and Languages**, Palm Programming, **The Developer's Guide**, Ed., O'Reilly, 1999.

[ROM 02] ROMÁN, M., et al., **A middleware infrastructure for Active Spaces**. **IEEE Pervasive Computer**, 2002, v. 1, n. 4, p. 74-83.

[RUI 05] RUIZ, L. B., **Arquitetura para RSSFs**, Departamento de Ciência da Computação da UFMG, http://www.dcc.ufmg.br/~linnyer/, acessado em junho de 2005.

[SAL 99] SALBER, D., DEY, A.K., ABOWD, G.D., **The Context Toolkit**: Aiding the Development of Context-Enabled Applications. *CHI'99*, ACM Press, 1999, p.434-441, Pittsburgh, PA.

[SALUTATION] **Salutation**, http://www.salutation.org, acessado em maio de 2005.

[SCH 95] SCHREIBER, R., **Middleware demystified**. Datamation, Cahners Publishing Company, Londres, Reino Unido, 1995, v. 41, n. 6, p. 41-45, Abr 95. ISSN 0011-6963.

[SDCARD] **SD Card**, http://www..sdcard.org/, acessado em maio de 2005.

[SIG 01] SIG BLUETOOTH, **Specification of the Bluetooth System**, **Specification Volume**, 2001.

[SNO 00] SNOEREN, A. C. E BALAKRISHNAN, H., **An End-to-End Approach to Host Mobility**, Proceedings of the **6th Annual International Conference on Mobile Computing and Networking** (MOBICOM 2000), Boston, MA, 2000.

[SOAP] **SOAP Version 1.2**, http://www.w3.org/TR/soap/, acessado em abril de 2005.

[SOU 01] SOUZA, L.F.H., **Estudos de Modelos de Serviços de Middleware e Proposta de Extensões à Plataforma JAMP**. Dissertação de Mestrado, Universidade Federal de São Carlos, 2001.

[STA 02] STALLINGS, W., **Wireless Communication and Networks**, Ed. Prentice Hall, 2002.

[STE 97] STEVENS, W.R., **UNIX Network Programming**: Networking APIs: Sockets and XTI, Prentice Hall PTR, 1997.

[REFLECTION] **Reflection API**, http://java.sun.com/docs/books/tutorial/reflect/index.html, **The** acessado em novembro de 2006.

[RMI] **RMI**, http://java.sun.com/docs/books/tutorial/rmi/, acessado em maio de 2005.

[SUR 06] **Jini**, http://www.surrogate.jini.org, acessado em maio de 2006.

[TAN 02] TANENBAUM, A., S., **Distributed Systems**, Ed. Prentice Hall, 2002.

[TAN 03] TANENBAUM, A., S., **Computer Networks**, Ed. 4, Prentice Hall, 2004.

[TINYOS] http://today.cs.berkeley.edu/tos/, acessado em maio de 2005.

[TREO] **Treo smartphones**, http://web.palmone.com/products/smartphones/treo650/details.jhtml, acessado em maio de 2005.

[UML] **UML**, http://www.uml.org, acessado em maio de 2005.

[UPNP] **UpnP**, http://www.upnp.org, acessado em maio de 2005.

[XML 99] ECKSTEIN, R., **XML Pocket Reference**, Ed. O'Reilly, 1999.

[XML 05] **Extensible Markup Language (XML)**, http://www.w3.org/XML/, acessado em maio de 2005.

[WAL 99] WALDO, J., **The Jini Archtecture for Network-centric Computing**, **Communications of the ACM**, 1999, vol. 42, nº 7.

[WAN 95] WANT, R. et al. **The PARCTAB Ubiquitous Computitig Experiment**. Technical Report CSL-95-1, Xerox Palo Alto Research Center, 1995.

[W3Ca 03] W3C**, Web Services Description Language (WSDL)** Version 1.2, disponível em: www.w3.org/TR/wsdl12/, 2003.

[W3Cb 04] W3C, **Recommendation: SOAP** Version 1.2 Part 2: Adjuncts, //www.w3.org/TR/soap12-part2/, 2003.

[WEI 91] WEISER, M., **The Computer for the 21st Century**, Scientific American, 1991.

[WIL 04] WILSON G., OSTREM J., BEY C, DUGGER M. **Palm OS® Programmer's API Reference**, PalmSource Inc, 2004.

[WIMAX] **WiMax**, http://www.teleco.com.br/tutoriais/tutorialwimax/, acessado em maio de 2005.