Gustavo Medeiros de Araújo

Uma Infraestrutura para Integração entre Dispositivos Computacionais Heterogêneos Baseada na Especificação DPWS

Florianópolis - SC

2009

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gustavo Medeiros de Araújo

UMA INFRAESTRUTURA PARA INTEGRAÇÃO ENTRE DISPOSITIVOS COMPUTACIONAIS HETEROGÊNEOS BASEADA NA ESPECIFICAÇÃO DPWS

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

Orientador: Prof. Dr. Frank Augusto Siqueira

Florianópolis, fevereiro de 2009

UMA INFRAESTRUTURA PARA INTEGRAÇÃO ENTRE DISPOSITIVOS COMPUTACIONAIS HETEROGÊNEOS BASEADA NA ESPECIFICAÇÃO DPWS

Gustavo Medeiros de Araújo

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

_____

Prof. Dr. Frank Augusto Siqueira
Coordenador do Curso

Banca Examinadora

_____

Prof. Dr. Frank Augusto Siqueira (Orientador)

_____

Prof. Dr. Antônio Augusto Medeiros Fröhlich

_____

Prof. Dr. Lau Cheuk Lung

_____

Prof. Dr. Leandro Buss Becker

Agradecimentos

Esta sessão reservo para agradecer a todos que de alguma forma me ajudaram durante todo o período de estudos. Primeiramente, agradeço a Deus pela oportunidade a mim concedida para realização tão almejada dos estudos que se iniciaram no ano de 2007. Em segundo lugar, agradeço aos meus pais e irmãos que sempre me apoiaram em todo esse período. Um agradecimento especial devo à minha esposa Tula, que esteve junto comigo me amparando nos momentos mais difíceis e dividindo as pequenas alegrias no decorrer desse período. Também quero deixar um agradecimento ao meu cachorro, Fred, que todos os dias, independente se eu estivesse estressado ou não, ele me recebia com alegria.

Agradeço também ao professor Frank Augusto Siqueira que me orientou em todo o trabalho do curso e sempre esteve presente com sua experiência e paciência. Um muito obrigado também ao pessoal da secretaria, em especial à Verinha, que sempre esclareceu prontamente as minhas dúvidas desde antes do começo do curso. Desde já deixo um agradecimento a todos os membros da banca, que proporcionaram com seu tempo e experiência realizar uma melhora no trabalho proposto.

Gostaria de agradecer também aos amigos da empresa Cetil, pois foi nessa empresa que comecei o primeiro ano de pós-graduação, e me foi concedido poder sair do trabalho para fazer as matérias do curso. Também tenho muito a agradecer aos amigos da empresa Goods That Talk – GTT, o Thiago, o Gustavo Yugo e ao Guido, que me ajudaram com os equipamentos de RFID para que eu pudesse realizar os experimentos que enriqueceram o meu trabalho e também pela concessão de um horário especial para realização do meu projeto da pós-graduação.

Sumário

Índice de Figuras

Índice de Tabelas

Lista de Abreviaturas

ARM : Advanced RISC Machines

CORBA : Common Object Request Broker Architecture

DCOM : Distributed Component Object Model

DP : Discovery Proxy

DPWS : Devices Profile for Web Services

DPWS4J : Devices Profile for Web Services for Java

DSB : Device Service Bus

GEN2 : Generation 2

GPS : Global Positioning System

HCI : Host controller interface

HTTP : Hypertext Transfer Protocol

IDL : Interface Definition Language

IEEE : Institute of Electrical and Electronics Engineers

Java CDC : Connected Device Configuration

Java ME : Java Micro Edition

JINI : Java Intelligent Network Infrastructure

L2CAP : Logical Link Control and Adaptation Protocol

MSDA : Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments

NFC : Near Field Communication

OSGi : Open Service Gateway Initiative

PDA : Personal Digital Assistant

RAM : Random-access Memory

RFCOMM : Radio frequency communications

RFID : Radio Frequency Identification

RQL : Reader Query Language

SDP : Service Discovery Protocol

SOA : Service-Oriented Architecture

SOA4D : Service-Oriented Architecture for Devices

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

SSH : Secury Shell

SSL : Secure Sockets Layer

Sun SPOT : Sun Small Programmable Object Technology

TCP/IP : Transmission Control Protocol / Internet Protocol

TCS BIN : Telephony control protocol-binary

UDDI : Universal Description Discovery Integration

UDP : User Datagram Protocol

UPnP : Universal Plug and Play

VM : V*irtual Machine*

W3C : World Wide Web Consortium

WPAN : Wireless Personal Area Network

WS4D : Web Service for Device

WSDL : Web Services Description Language

WSUN : Web Services on Universal Networks

XML : Extensible Markup Language

Resumo

Serviços computacionais têm proporcionado facilidades à vida das pessoas, provendo mais agilidade em operações e acesso a informação. Muitos desses serviços são providos por dispositivos que estão espalhados no ambiente de convívio comum. Alguns destes equipamentos podem se conectar e trocar dados entre si utilizando uma infra-estrutura de rede. Entretanto, um conjunto maior de serviços poderia ser oferecido se dispositivos que empregam diferentes tecnologias de comunicação também pudessem se conectar e trocar dados. Dessa forma, seria possível aproveitar classes diversas de equipamentos para construção de soluções que atendam cenários de aplicação diversos.

Com o objetivo de proporcionar um mecanismo que reúna serviços oferecidos por dispositivos heterogêneos é proposto neste trabalho o *Device Service Bus* (DSB). O DSB consiste em uma infra-estrutura de *middleware* empregada para integração de diferentes dispositivos embarcados em ambiente de computação ubíqua. Baseado no *Devices Profile for Web Services* (DPWS) como tecnologia de integração, o DSB permite interação de vários dispositivos que adotam diferentes padrões de comunicação.

Esse trabalho apresenta uma implementação de um protótipo do DSB, que provê componentes de software que são implantados em dispositivos responsáveis por construir um barramento de integração entre dispositivos heterogêneos. Além disso, dois cenários de aplicação real reunindo sensores Sun SPOT e leitores RFID, e medições de desempenho obtidas com estes dispositivos também são apresentados nesse trabalho.

Abstract

Computing services have provided facilities to people´s lives, providing more agility to operations and information access. Several of these services are provided by devices that are spread in the human environment. Some of these devices can connect to each other and exchange data using a network infrastructure. However, a bigger amount of services could be offered if equipmentsthat employ different communication technologieswere also able to interconnect and exchange data. This would allow the use of the mostdiverse classes of devices to build solutions targeting a wide variety of application scenarios.

In order to provide a mechanism able to put together all services provided by different devices, this work proposes the Device Service Bus (DSB). The DSB consists in a middleware infrastructure employed for integration of heterogeneous embedded devices in ubiquitous computing environments. Based on the Devices Profile for Web Services (DPWS) as the underlying integration technology, the DSB allows the interaction among devices that adopt different networking standards.

This work presents a prototype implementation of the Device Service Bus, which provides software components that are deployed on devices responsible for building an integration bus among heterogeneous devices. Moreover, two real-world applications scenarios with Sun SPOT sensors and RFID readers, and performance measurements obtained with these devices are also presented in this work.

1. Introdução

A computação tem evoluído ao ponto de tornar cada vez mais presentes recursos tecnológicos no cotidiano das pessoas. Tanto no trabalho, em lugares públicos e em casa, os mais diversos tipos de dispositivos computacionais são empregados com intuito de facilitar a vida do homem. Com a constante evolução dos dispositivos computacionais, novos desafios surgem a cada dia em relação à integração de tais dispositivos, que se faz necessária para permitir a execução conjunta de tarefas computacionais. A crescente tendência de adoção destes equipamentos, que vão desde PDAs (*Personal Digital Assistant*) e *SmartPhones* até dispositivos com recursos computacionais mais limitados, como sensores e *tags* RFID, aumenta a disponibilidade de serviços de *softwares* e a troca de informações.

Muitos dispositivos não se beneficiam de uma infra-estrutura de rede estável, como ocorre com outras aplicações distribuídas. Por outro lado, esses dispositivos dependem de protocolos de rede *ad-hoc* e não têm conhecimento prévio de outros equipamentos conectados à rede e nem dos serviços que os mesmos possam prover. Como dispositivos podem entrar e sair da rede de modo imprevisível, não há garantias da disponibilidade de serviços que estejam em execução. Tais restrições estão presentes em um cenário que foi definido por Mark Weiser como ambiente de computação ubíqua [Weiser 1993].

Com a evolução das redes sem fio e dos dispositivos móveis, outros padrões para comunicação em rede têm surgido, atendendo a diferentes aplicações e classes de dispositivos. A integração de diferentes classes de dispositivos que utilizam redes heterogêneas é ainda uma área aberta à pesquisa.

A Arquitetura Orientada a Serviços (SOA) [ERL 2004] tem se mostrado uma solução promissora para integração de sistemas heterogêneos. SOA consiste em um paradigma que permite a construção de componentes de software com baixo acoplamento, que provê serviços que podem ser localizados dinamicamente e invocados por clientes usando um protocolo de comunicação conhecido pelas partes envolvidas.

Serviços Web consistem em uma tecnologia popular para construção de software baseado na arquitetura SOA. A tecnologia de Serviços Web adota padrões largamente disponíveis para representação e comunicação de dados, basicamente XML, HTTP e SOAP. A adoção desses padrões permite a implantação de serviços Web em virtualmente qualquer ambiente computacional [W3C 2004], habilitando dessa forma a interação entre provedores de serviço (i.e servidores) e consumidores (clientes) em um ambiente heterogêneo de computação.

SOA e Serviços Web têm sido adotados com sucesso em ambiente de negócio, [VINOSKI, 2003] onde diferentes sistemas de informação aplicados para construção de regras de negócio têm se transformado em serviços, facilitando a comunicação e a troca de informações entre sistemas. Essa integração de sistemas permite que as corporações redirecionem o foco no desenvolvimento de processos de negócio, ao invés de gastar recursos na operação e manutenção de sistemas. Apesar de ser o cenário mais comum no qual SOA é usada atualmente, este não é o único cenário na qual SOA é capaz de prover integração entre *softwares* [Machado 2006] [Schall, Aiello e Dustdar 2006].

1.1 Motivação

Durante os últimos anos, houve um grande aumento na fabricação de dispositivos programáveis – como telefones móveis, PDAs, sensores, maquinário industrial, e equipamentos médicos. A capacidade de processamento de tais equipamentos têm evoluído,e estes vêm ganhado novas funcionalidades, que incluem a capacidade de comunicação entre pares utilizando redes sem fio e também de executar protocolos de rede de alto nível que são requeridos pela arquitetura SOA.

Apesar do rápido crescimento na produção e venda de dispositivos embutidos, a falta de adoção de padrões comuns de comunicação e para representação de dados torna difícil e muitas vezes inviável a interação entre os dispositivos.

Em virtude da necessidade de integração de diversos dispositivos, independentemente de plataforma e de mecanismos de comunicação, foi proposta a especificação *Device Profile for Web Service* (DPWS) [Microsoft 2006]. DPWS é baseado na arquitetura SOA e utiliza os mesmos protocolos dos Serviços Web ,como HTTP, TCP, SOAP, XML além de UDP (*multicast* e *unicast*), permitindo a integração de dispositivos em redes *ad-hoc*. Com base nessa especificação, é possível criar uma infra-estrutura de *middleware* visando prover um barramento de comunicação independente de tecnologias para integração de diferentes tipos de dispositivos computacionais.

1.2 Objetivos

Nesta sessão serão apresentados os objetivos do trabalho proposto, sendo divididos em objetivo geral e específicos.

1.2.1 Objetivo Geral

Objetivo geral deste trabalho é prover uma infra-estrutura de *middleware* que possibilite a interoperabilidade de dispositivos computacionais embarcados que adotem diferentes tecnologias de comunicação.

1.2.2 Objetivos Específicos

Com a realização deste trabalho, espera-se que dispositivos que utilizem diferentes tecnologias de comunicação, como Leitores RFID [Want 2006], dispositivos Bluetooth [Chatschik 2001] e sensores Sun SPOT [Sun 2004], possam interagir e disponibilizar seus serviços na rede na forma de Serviços Web.

Os objetivos específicos são:

- Prover vários pontos de acesso ao *middleware*, podendo ser implantado tanto em estações de trabalho como em equipamentos com recursos computacionais limitados como PDA ou *set-top box*.
- Prover acesso simultâneo a seus metadados e suas descrições de serviços, sendo disponibilizados por meio de uma interface padrão como o WSDL.
- Permitir que dispositivos possam ser anunciados quanto a sua entrada em uma rede e também quanto a sua saída da rede, indicando

indisponibilidade do serviço. Além disso, que tais dispositivos possam procurar por outros dispositivos e serviços na rede e executar as suas ações.

1.3 Justificativa

A interação entre dispositivos computacionais é uma realidade que desperta interesse tanto para o meio acadêmico quanto para as indústrias. Tornar equipamentos de diferentes fabricantes disponíveis em ambiente de rede e prover meios para que tais dispositivos possam interagir levaram ao desenvolvimento do *Device Service Bus* (DSB), que é baseado na especificação DPWS.

Sendo o núcleo da solução escrito com base em uma especificação aberta, empregando padrões e protocolos de rede largamente utilizados, como HTTP, TCP, UDP (*multicast* e *unicast*), SOAP, e XML, mostrou-se uma solução viável para prover interconexão entre dispositivos heterogêneos.

Além disso, a especificação DPWS é norteada para o desenvolvimento de aplicações seguindo o paradigma SOA. Por mais essa característica, notou-se que trazer para o ambiente da computação embarcada a disponibilização de serviços, criando atores consumidores e provedores, o qual já são utilizados largamente no mundo corporativo, é uma oportunidade para representação de maneira homogênea dos dispositivos computacionais presentes em um ambiente ubíquo e heterogêneo.

1.4 Metodologia

Primeiramente, será estudada a especificação DPWS e analisados *frameworks* que implementem a especificação. Ademais, serão estudados os trabalhos relacionados na área de interoperabilidade de dispositivos, a fim de aprender com esses esforços e procurar por oportunidades de evolução para o desenvolvimento do projeto de pesquisa.

Em seguida, serão estudadas tecnologias como *Radio Frequency Identification* (RFID), sensores Sun SPOT, o padrão de comunicação *Bluetooth* e aplicações integrando essas tecnologias em casos de uso reais.

O estudo destas tecnologias tornará possível realizar implementações e experimentos com o barramento de comunicação que será criado. Medições serão realizadas para cada tecnologia com intuito de mensurar o tempo de resposta de descobrimento dos dispositivos e serviços e o tempo de resposta da invocação desses serviços por meio da solução proposta. Também será analisada a capacidade de lidar com vários dispositivos conectados ao barramento.

1.5 Resultados Esperados

Após a realização deste trabalho espera-se criar um canal de comunicação independente de plataforma utilizando protocolos padrões de comunicação da Internet para disponibilizar diferentes classes de dispositivos. Dessa forma, a possibilidade de interconectar tecnologias com padrões heterogêneos pode gerar um reaproveitamento das tecnologias e padrões de comunicação diversos.

1.6 Estrutura do Documento

Os próximos capítulos do documento serão apresentados da seguinte forma:

O segundo capítulo apresentará a revisão bibliográfica. Conceitos importantes para compreensão do trabalho serão introduzidos. Os trabalhos relacionados na área de interoperabilidade de equipamentos serão discutidos.

O terceiro capítulo detalha a proposta de trabalho, apresentando a arquitetura da solução e a sua implementação em casos de uso.

O quarto capítulo mostra os resultados dos experimentos realizados e a análise dos testes obtidos com a plataforma de teste composta por diferentes dispositivos.

O quinto capítulo apresenta a conclusão do trabalho e sua aderência à proposta realizada, e também expressa as perspectivas futuras para continuação do trabalho.

2. Revisão Bibliográfica

Este capítulo disserta a respeito dos principais conceitos que envolvem o trabalho proposto. Serão descritos os dispositivos embarcados, dando ênfase às tecnologias de comunicação por eles utilizadas. Será abordada também a arquitetura SOA, juntamente com tecnologia de serviços Web, e a especificação DPWS. Por fim, serão apresentados alguns esforços de pesquisa na área de interoperabilidade de dispositivos embarcados.

2.1 Computação Embarcada

A computação embarcada (também conhecida como computação embutida) consiste na execução de um projeto integrado de *hardware* e *software* para realizar funções específicas, geralmente com algumas restrições, como tempo real. Como a computação embarcada é projetada para execução de tarefas específicas, é possível ter desde microprocessadores a super-computadores com sistemas embarcados responsáveis por realizar processamentos, como monitoramento de temperatura em um sensor ou controle de tráfego aéreo. Dessa forma, pode-se ter dispositivos com restrições de recursos, como telefones celulares executando um conjunto de funções específicas..

Os projetos de dispositivos embarcados estão presentes em várias aplicações comerciais e industriais. PDAs, celulares, set-top boxes e sensores são exemplos de dispositivos embarcados. Segundo [WOLF, 2001] os dispositivos embarcados devem prover as seguintes funcionalidades básicas:

- Algoritmos complexos: os microprocessadores podem executar tarefas críticas e complexas como, por exemplo, um motor de automóvel deve saber filtrar a quantidade de combustível de acordo com o desempenho exigido e ao mesmo tempo deve controlar a emissão de poluentes no meio ambiente.
- Interface com usuário: os microprocessadores podem atuar como controladores de interfaces complexas com múltiplas opções de menus ou movimento de mapas para sistemas de navegação GPS (*Global Positioning System*).
- Tempo Real: muitos sistemas computacionais embutidos devem executar tarefas em tempos determinados. A falta de precisão temporal na execução de determinadas tarefas podem causar falhas graves no sistema e, dependendo da aplicação, pode colocar em risco vidas humanas.

Algumas tecnologias de comunicação foram desenvolvidas para ampliar as funcionalidades dos sistemas embarcados e para proporcionar conectividade a dispositivos. Tecnologias como Bluetooth [Chatschik 2001] , *Radio Frequency Identification* (RFID) [Want 2006] e redes de sensores [Loureiro, Nogueira, Ruiz, et al 2004], são exemplos de tecnologias que permitem trocas de dados entre dispositivos com restrições de recursos computacionais. A seguir serão apresentados mais detalhes a respeito dessas tecnologias.

2.1.1 *Bluetooth*

Bluetooth é uma especificação aberta criada por um consórcio de empresas que tem como objetivo criar redes sem fio de comunicação pessoal (*Wireless Personal Area*

Network - WPAN). *Bluetooth* foi projetado para ter alcance de 10 a 100 metros, provendo conexão e troca de informação entre dispositivos. Outra característica de projeto é o fato da comunicação gerada pela tecnologia *Bluetooth* consumir poucos recursos computacionais. Isso se deve ao fato da plataforma-alvo para implementação do *Bluetooth* ser composta por dispositivos com recursos limitados, com celulares e PDAs, dentre outros.

A figura 2.1 mostra a pilha de protocolo *Bluetooth* que apresenta os seguintes componentes:

- *BaseBand*: é o componente responsável pelo estabelecimento de conexão com uma rede piconet, definir o formato do pacote de transmissão,endereçamento, temporização e controle de energia.
- *Link Manager*: é o componente responsável por estabelecer a configuração do *link* entre dispositivos Bluetooth, gerenciar os *links* correntes e aspectos de segurança como autenticação e encriptação.
- *Host controller interface* (HCI): é o componente que fornece um método interface uniforme para acessar recursos de hardware Bluetooth.
- *Logical Link Control and Adaptation Protocol (L2CAP)*: componente responsável por adaptar os protocolos superiores a camada *baseband*, fornecendo tanto serviços sem conexão como serviços orientado a conexão.
- *Telephony control protocol-binary (TCS BIN)*: é o componente que define o controle de chamada de sinalização para estabelecimento de chamadas de voz e dados entre dispositivos *Bluetooth*.
- *Radio frequency communications (RFCOMM)*: é o componente que simula conexão serial RS-232.

- *Service Discovery Protocol (SDP)*: é o componente responsável por gerenciar informações dos dispositivos, a fim de realizar descobrimento de dispositivos *Bluetooth.*

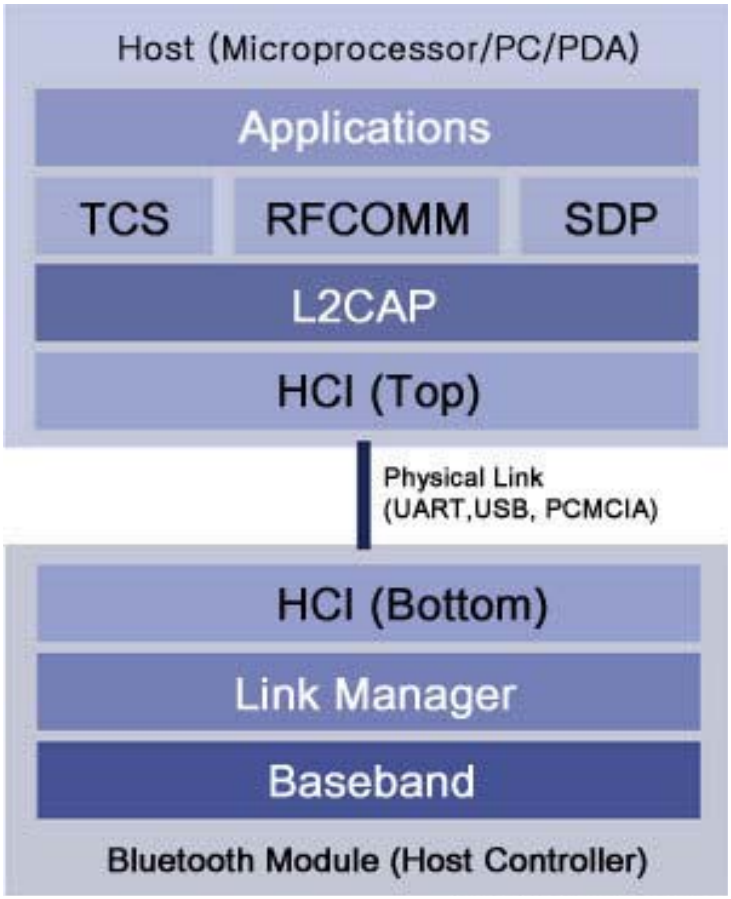

Figura 2.1. Pilha de Protocolo Bluetooth

Dois tipos de redes podem ser criados com a tecnologia Bluetooth: as *piconets* e *scatternet*. A Figura 2.2 ilustra a formação dessas redes. Quando dois ou mais dispositivos se conectam é formada uma rede *piconet,* que pode ser formada por até oito dispositivos. O dispositivo que iniciou a conexão é chamado de *master* e os outros são os *slaves.* O dispositivo *master* da conexão deve regular a taxa de transmissão de dados e a sincronização entre os pares.

Quando dispositivos de redes *piconets* diferentes se conectam, é criada uma rede *scatternet*. Os dispositivos que se conectam a mais de uma rede *piconet* são os *slaves*, pois os dispositivos *master* só podem se conectar a uma rede *piconet*.

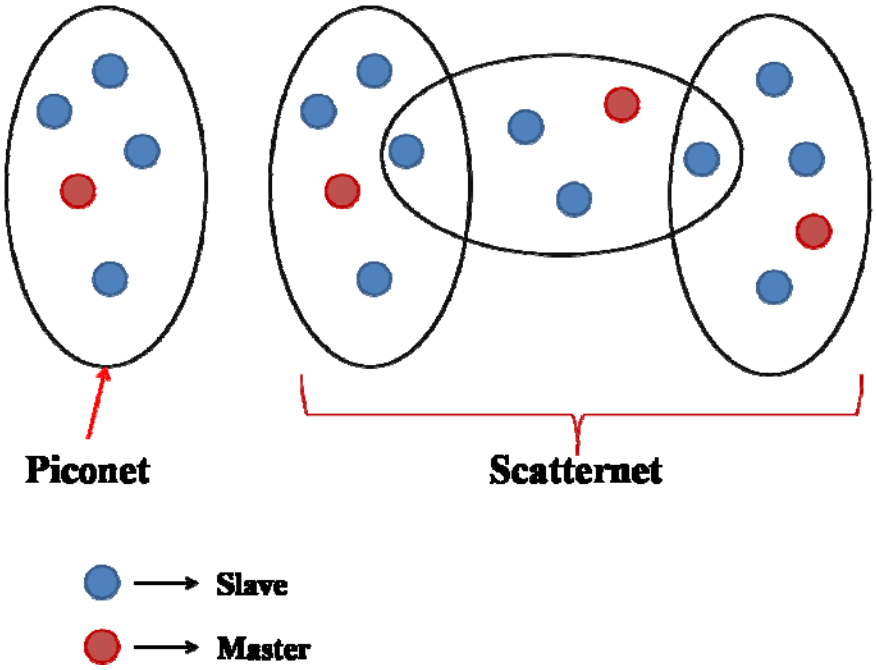

Figura 2.2. Redes *Bluetooth*

2.1.2 *Radio Frequency Identification* - RFID

A tecnologia de identificação por rádio freqüência trata-se de um mecanismo de identificação automática por meio de envio de sinais de rádio, recuperando e armazenando dados remotamente em dispositivos chamados de *tags* ou etiquetas.

As etiquetas, como mostra a Figura 2.3, são *transponders* que possuem um *chip* e uma antena que possibilita responder por sinais de rádio enviados por transmissores

que são Leitores RFID [Fuhrer, Guinard e Liechti 2006]. Existem basicamente três tipos de etiquetas, são elas:

- Etiquetas passivas: funcionam sem bateria interna, sendo possível realizar escrita e leitura do conteúdo do *chip*. A alimentação para operacionalizar a etiqueta passiva é fornecida pelo Leitor RFID quando este transmite um sinal. O sinal carrega a etiqueta fazendo com que ela processe a informação requerida no *chip* e responda ao Leitor com informações armazenadas.
- Etiquetas semi-passivas: funcionam com bateria própria, permitindo realizar escrita e leitura. Têm maior capacidade de armazenamento de dados que a etiqueta passiva e possuem circuito integrado. As operações são realizadas de maneira passiva, ou seja, ela só envia sinal quando recebe um sinal de requisição.
- Etiquetas ativas: funcionam com bateria própria, e possuem uma infra-estrutura de hardware mais elaborada, podendo realizar conexões *Ethernet*. Possuem maior capacidade de armazenar dados do que as etiquetas passivas e semi-passivas. Podem enviar sinais para outras etiquetas ativas e para os leitores.

Figura 2.3. Etiquetas RFID

As etiquetas RFID podem ser utilizadas em várias aplicações com intuito de prover rastreabilidade e identificação automática de produtos. Etiquetas passivas podem ser aplicadas a passaportes, cartões de crédito e qualquer tipo de documento, enquanto etiquetas semi-passivas ou ativas podem ser usadas para rastrear contêineres e automóveis, por exemplo.

Os leitores RFID, como mostra a Figura 2.4, são dispositivos responsáveis por realizar a leitura do conteúdo das etiquetas RFID e escrita de dados na memória das etiquetas. Os leitores RFID são dotados de protocolos anti-colisão para conseguir ler e distinguir unicamente cada etiqueta. A distância que é possível ler dados das etiquetas varia de acordo com o dispositivo leitor e suas antenas, podendo ser de poucos centímetros a metros de distância. As leituras de curta distância, chamadas de *Near Field Communication* (NFC), geralmente são utilizadas em aplicações de controle de acesso.

Figura 2.4. Leitor RFID

Os dispositivos leitores RFID podem ser conectados a uma infra-estrutura computacional, dessa forma outros sistemas podem utilizar tais dispositivos. Os leitores RFID provêem interfaces de comunicação com o meio externo por meio de interfaces de rede *Ethernet* ou Bluetooth, ou por conexão via USB ouRS-232. Os sistemas podem enviar comandos hexadecimais ou na linguagem de consulta RQL (*Reader Query Language*) para realizar tarefas de leitura e escrita, podendo variar o tempo de leitura e realizar filtragem de dados.

2.1.3 Redes de Sensores

Sensores são dispositivos que medem condições de ambiente e converte as medições em sinais que podem ser lidos por instrumentos. Com o avanço da tecnologia de microprocessadores, os sensores foram dotados de mecanismos que lhes proporcionaram maior processamento de informações e comunicação de dados

[Loureiro, Nogueira, Ruiz, et al 2004]. Dessa forma, redes de sensores puderam ser criadas no intuito dos dispositivos sensores trocarem informações e serem integrados a sistemas de informação.

Algumas aplicações podem ser desenvolvidas com o uso de redes de sensores. Aplicações de segurança podem ser dotadas de sensores que monitorem imagens, perturbações acústicas, sendo que cada sensor pode realizar um tipo de medição e todos podem ser integrados a um terceiro *software* que realize o gerenciamento das medições realizando alertas aos usuários. Aplicações para monitorar o ambiente no interior de prédios e ambientes como florestas também podem utilizar sensores para detectar a elevação de temperatura com o objetivo de alertar possibilidade de incêndio. Outras aplicações podem utilizar sensores em equipamentos médicos para monitorar alterações de substâncias químicas no interior do organismo do paciente, assim, enviando alertas às equipes médicas.

Existem sensores, como os ilustrados na Figura 2.5, que permitem mais de um mecanismo de medição, como medição de temperatura, intensidade de luz, movimento e aceleração. O sensor Sun SPOT [Sun 2004], desenvolvido pelo laboratório da empresa Sun Microsystems, agrega várias funcionalidades de medições de condições de ambiente além de realizar comunicação entre outros sensores Sun SPOT e prover interface de comunicação com ambiente externo.

O padrão de comunicação utilizado pelo Sun SPOT é o IEEE 802.15.4 [Howitt e Gutierrez 2003] que também é base para especificações como ZigBee [Zigbee™ Alliance]. O padrão IEEE 802.15.4 especifica camada de rede do tipo WPAN (*Wireless Personal Area Network*), que tem como objetivo otimizar recursos computacionais, provendo uma baixa taxa de transferência de dados.

Figura 2.5. Sensores

2.2 Arquitetura Orientada a Serviços

O paradigma da Arquitetura Orientada a Serviços (SOA) torna componentes distribuídos em serviços que possam ser acessados por meio de um protocolo comum [Erl 2004]. Nesta arquitetura, serviços são disponibilizados por provedores de serviço (*Service Providers*) e a interface dos serviços pode ser publicada em um repositório (*Service Broker*). Dessa forma, as interfaces ficam disponíveis para clientes (*Service Consumers*) pertencentes ao ambiente de rede. Clientes podem realizar buscas por serviços no repositório e, com a posse da interface do serviço e do seu endereço, o serviço pode ser invocado pelo cliente, como mostra a Figura 2.6. Serviços no paradigma SOA têm baixo acoplamento e granularidade grossa, e a característica de poderem ser acessados em diversos escopos de rede, observadas as devidas permissões.

Várias tecnologias, como Serviços Web, *Salutation*, *Service Location Protocol*, Bonjour [Edwards 2006], implementam o conceito SOA, fornecendo mecanismos para acesso, descoberta e ligação ao serviço.

Figura 2.6 Arquitetura Orientada a Serviço

2.3 Serviços Web

A tecnologia de Serviços Web permite que aplicações interajam em ambiente de rede local e de longa distância empregando os protocolos HTTP, SMTP e TCP/IP. Serviços Web realizam busca, associação e invocação sobre o protocolo SOAP [ROY; RAMANUJAN, 2001], permitindo a criação e interação de atores provedores de serviços (servidor) e consumidores de serviços (cliente).

As principais características da tecnologia de Serviços Web são [CHAPPELL; JEWELL, 2002]:

- SOAP - Protocolo independente da camada de transporte baseado em XML.
- WSDL - Descrição de serviço baseado em XML.

- UDDI - Repositório de Descrições de Serviço que permite descoberta distribuída.
- Permite invocações síncronas, assíncronas e baseadas em eventos.
- Permite interações por chamada remota de procedimento (RPC).
- Provê suporte a transações seguras utilizando criptografia.
- Permite anexar objetos serializados para realizar troca de documentos.

As interações entre os serviços da tecnologia de Serviços Web são demonstrados na Figura 2.7, a qual segue a arquitetura SOA. A seguir são descritos os principais elementos desta tecnologia:

- O *Universal Description, Discovery and Integration* (UDDI) é uma especificação para construção de um diretório distribuído de serviços na Web. O UDDI armazena informações em um formato XML específico – o WSDL – e também permite buscar dados no repositório.
- O protocolo SOAP, padronizado pela W3C, permite a comunicação independente do protocolo de transporte e plataforma.
- *Web Service Description Language* (WSDL) é a linguagem de descrição dos serviços Web. Especifica a interface do serviço Web e é independente de linguagem de implementação. WSDL é equivalente à linguagem IDL nas arquiteturas CORBA ou DCOM.
- O cliente é o consumidor do serviço Web. Para se tornar um consumidor do serviço Web o cliente deve saber processar respostas e requisições SOAP.

Figura 2.7. Interação de Serviço Web

2.4 A Especificação *Devices Profile for Web Services*

O *Devices Profile for Web Services* é uma pilha de protocolos que provê mecanismos de comunicação de alto nível para interoperabilidade entre dispositivos [Microsoft 2006]. Seguindo os padrões utilizados pelos Serviços Web, DPWS une o cenário de sistemas embarcados com o mundo de aplicações baseadas em SOA, proporcionando às aplicações embarcadas o mesmo nível de interoperabilidade disponível para sistemas de informação. A pilha de protocolos, apresentada na Figura

2.8, demonstra como o DPWS emprega protocolos padrão da Internet, como TCP ,UDP *unicast* e *multicast*, e HTTP. Para troca de mensagens, o protocolo SOAP é empregado tanto sobre UDP quanto sobre HTTP.

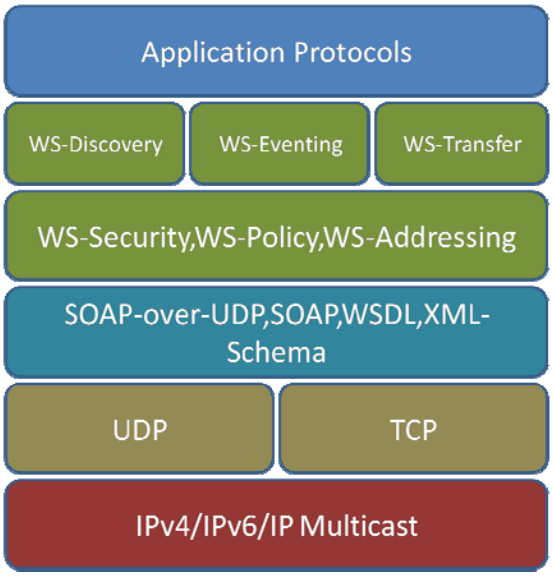

Figura 2.8 Protocolos do DPWS

DPWS proporciona a construção de dois tipos de serviços:

- *Hosting Service*: Serviço que hospeda outros serviços e recebe diferentes tipos de mensagens. Em geral representa o dispositivo no qual o serviço está hospedado.
- *Hosted Service*: serviço hospedado por um *hosting service*, com o tempo de vida limitado ao tempo de vida do *hosting service*. *Hosted services* são visíveis na rede e são endereçados independentemente do *hosting service*.

Como mostra a Figura 2.8, os principais protocolos padrão utilizados pelos serviços Web também são adotados pela especificação DPWS. Esses padrões são:

- *WS-Addressing*: encapsula todas as informações de endereçamento de rede em campos no cabeçalho da mensagem, como “To:” e “Reply to:”. Isso permite ao protocolo SOAP ser independente do protocolo de transporte.
- *WS-MetadataExchange*: provê acesso dinâmico aos metadados que descrevem o *hosted* e *hosting service*.
- *WS-Discovery*: define um mecanismo de descobrimento de dispositivos, baseado em mensagens *multicast* para localização de serviços disponíveis na rede. Para estender o descobrimento para outras redes, o padrão define o *Discovery Proxy*, que reduz o tráfego de mensagens geradas pelo uso do protocolo *multicast*.
- *WS-Policy*: provê um modo de adicionar informações à descrição WSDL, que são especificadas na forma de políticas suportadas pelos serviços.
- *WS-Eventing*: define um conjunto de operações para notificação de eventos, permitindo que serviços publiquem e recebam mensagens assíncronas.
- *WS-Security*: provê mecanismos de segurança para interação entre equipamentos, garantindo propriedades de segurança como autenticação, confidencialidade e integridade.

As mensagens definidas pela especificação DPWS, que são trocadas entre os dispositivos durante o processo de descobrimento, são mostradas na Figura 2.9. A mensagem *Hello* é um anúncio *multicast* realizado pelo dispositivo quando entra na rede. A mensagem *multicast Probe* é enviada por um cliente quando busca por um *Hosting Service* (i.e. dispositivos) com características particulares descritas no corpo da mensagem. Um ou mais dispositivos que corresponderem às características procuradas respondem com a mensagem *ProbeMatch*, contendo o metadado no qual os *hosted services* são listados. A descrição dos serviços disponíveis pode ser obtida enviando a

mensagem *Get*, que resulta na mensagem *GetResponse*, que é retornada ao cliente contendo a descrição do serviço requerido.

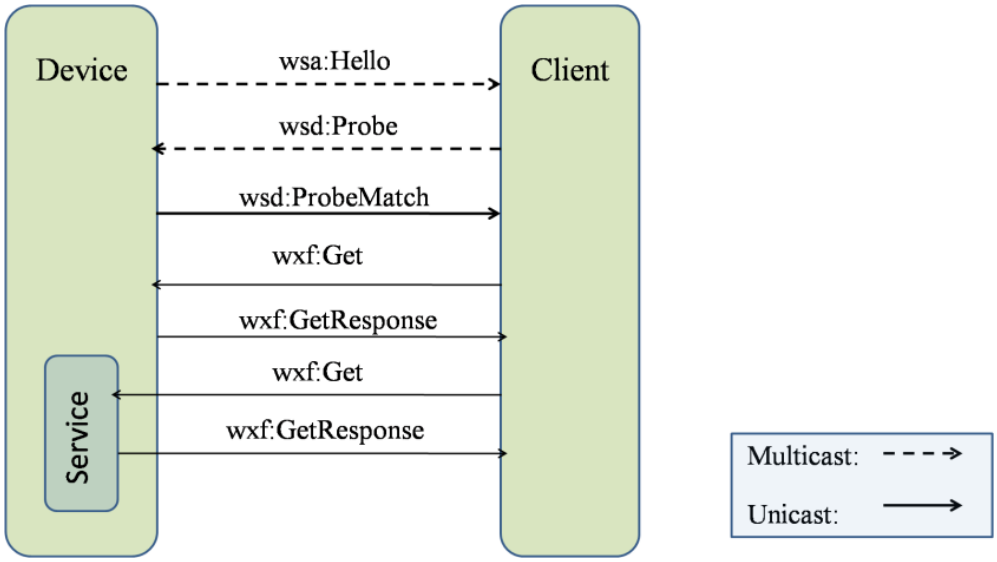

Figura 2.9. Mensagens do DPWS

Um componente importante do DPWS é o *Discovery Proxy* que é responsável por ampliar a área de descoberta de dispositivos e serviços e interação entre eles. O descobrimento do DPWS é baseado em mensagens *multicast*, por isso está limitado a redes locais. Para sanar essa limitação o *Discovery Proxy* (DP) foi projetado para ser descoberto por serviços DPWS e serem usados para lidar com os mecanismos de mensagens *multicast* e *unicast*. Dessa forma o DP introduz duas funções: reduzir o tráfego de mensagens *multicast* e estender o alcance de rede para o descobrimento de serviços para dispositivos localizados em uma rede local. A figura 2.10 ilustra o funcionamento do *Discovery Proxy*.

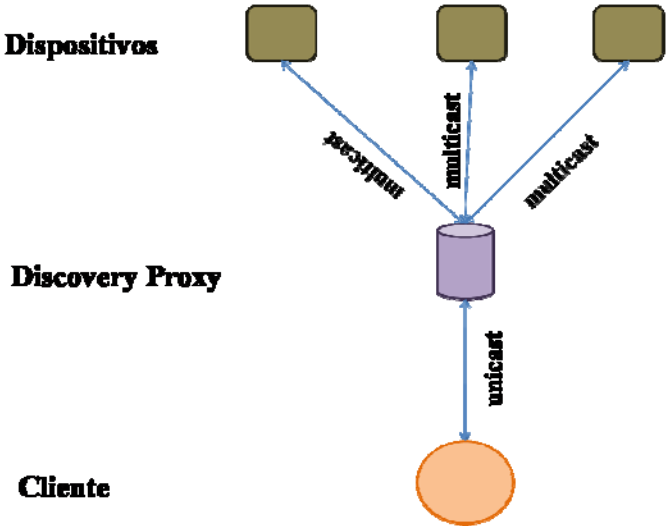

Figura 2.10 . Interação com *Discovery Proxy*.

Os metadados trocados entre os dispositivos durante o processo de descoberta, que descrevem o *hosting service* e os *hosted services*, são divididos em quatro sessões:

- *ThisModel*: descreve características que são comuns a um grupo de dispositivos que pertencem à mesma classe, como nome e URL do fabricante, código e nome do modelo.
- *ThisDevice*: descreve características que podem ser diferentes de um dispositivo para outro, como número serial , *friendlyname* e versão do *firmware*.
- *Relationship*: descreve a relação entre os serviços, dando detalhes sobre o *hosting service* e seus *hosted services*.
- *WSDL*: descreve todas as operações, faltas e estruturas de dados suportadas por um *hosted service* em particular.

As Figuras de 2.11 a 2.14 ilustram as partes das sessões de metadados que definem a descrição da interface do dispositivo e dos seus serviços.

```
<wsdp:ThisModel
    xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof" >
  <wsdp:Manufacturer>ACME Manufacturing</wsdp:Manufacturer>
  <wsdp:ModelName xml:lang="en-GB" >ColourBeam 9</wsdp:ModelName>
  <wsdp:ModelName xml:lang="en-US" >ColorBeam 9</wsdp:ModelName>
</wsdp:ThisModel>
```

Figura 2.11. Sessão *ThisModel*

```
<wsdp:ThisDevice
    xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof" >
 <wsdp:FriendlyName xml:lang="en-GB" >
   ACME ColourBeam Printer
 </wsdp:FriendlyName>
 <wsdp:FriendlyName xml:lang="en-US" >
   ACME ColorBeam Printer
 </wsdp:FriendlyName>
</wsdp:ThisDevice>
```

Figura 2.12. Sessão *ThisDevice*

```
<wsdp:Relationship
    Type="http://schemas.xmlsoap.org/ws/2006/02/devprof/host"
    xmlns:img="http://printer.example.org/imaging"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof" >
  <wsdp:Hosted>
    <wsa:EndpointReference>
      <wsa:Address>http://172.30.184.244/print</wsa:Address>
    </wsa:EndpointReference>
    <wsdp:Types>
      img:PrintBasicPortType img:PrintAdvancedPortType
    </wsdp:Types>
    <wsdp:ServiceId>
      http://printer.example.org/imaging/PrintService
    </wsdp:ServiceId>
  </wsdp:Hosted>
</wsdp:Relationship>
```

Figura 2.13. Sessão *Relationship*

```
<wsx:MetadataSection Dialect="http://schemas.xmlsoap.org/wsdl">
      <wsdl:definitions targetNamespace="www.ppgcc.inf.ufsc.br"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="www.ppgcc.inf.ufsc.br"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/">
        <wsdl:types>
          <xs:schema targetNamespace="www.ppgcc.inf.ufsc.br" xmlns:xs="http://www.w3.org/2001/XMLSchema">
            <xs:element name="retorno" type="xs:string" />
          </xs:schema>
        </wsdl:types>
        <wsdl:message name="ReadinputMsg" />
        <wsdl:message name="ReadoutputMsg">
          <wsdl:part name="retorno" element="tns:retorno" />
        </wsdl:message>
        <wsdl:portType name="ReadTags">
          <wsdl:operation name="Read">
            <wsdl:input message="tns:ReadinputMsg" wsa:Action="www.ppgcc.inf.ufsc.br/ReadTags/ReadRequest"
/>
            <wsdl:output message="tns:ReadoutputMsg"
wsa:Action="www.ppgcc.inf.ufsc.br/ReadTags/ReadResponse" />
          </wsdl:operation>
        </wsdl:portType>
        <wsdl:binding name="ReadTagsBinding" type="tns:ReadTags">
          <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
          <wsdl:operation name="Read">
            <soap:operation soapAction="www.ppgcc.inf.ufsc.br/ReadTags/ReadRequest" />
            <wsdl:input>
              <soap:Body use="literal" />
            </wsdl:input>
            <wsdl:output>
              <soap:Body use="literal" />
            </wsdl:output>
          </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="ReadTagsService">
          <wsdl:port binding="tns:ReadTagsBinding" name="ReadTagsPort">
            <soap:address location="http://127.0.0.1:26653/ReadTags" />
          </wsdl:port>
        </wsdl:service>
      </wsdl:definitions>
    </wsx:MetadataSection>
```

Figura 2.14. Sessão WSDL

2.5 Outras Tecnologias de Integração

Algumas tecnologias de integração, como *Universal Plug and Play* (UPnP) [UPnP Forum 2006], *Java Intelligent Network Infrastructure* (JINI) [Sun 2005] e *Open Service Gateway Initiative* (OSGi) [OSGi 2005] foram propostas com o objetivo de padronizar a comunicação e mecanismos de localização de serviços, permitindo interação entre equipamentos. JINI provê uma solução completa para permitir interoperabilidade entre dispositivos, entretanto é limitada à plataforma Java. UPnP adota padrões de implementação de Internet bem conhecidos, como HTTP, SOAP e XML, para comunicação entre dispositivos, mas sua adoção é limitada a redes locais. A

especificação OSGi define uma plataforma robusta e segura de serviços baseada em Java, dessa forma limitando-se a essa plataforma.

A tabela 1 [Bohn, Bobek e Golatowski 2006] ilustra uma comparação de características como segurança, escalabilidade, mecanismo *plug and play*, suporte a dispositivos dentre outras características, entre tecnologias para integração entre serviços. Pode-se notar que a especificação DPWS cobre vários aspectos importantes para a interoperabilidade entre dispositivos.

	OSGi	**JINI**	**UPnP**	**Web Services**	**DPWS**
Plug and Play	-	X	X	-	X
Suporte a Dispositivo	X	X	X	-	X
Independência de Plataforma	-	-	X	X	X
Independência de Mecanismos de rede	-	X	X	X	X
Alta escalabilidade	X	X	-	X	X
Segurança	X	X	-	X	X
Aceitação de Mercado	X	X	X	X	X

Tabela 1. Comparação das características das tecnologias de integração

Diante dos dados apresentados na Tabela 1 sobre as tecnologias de integração, pode-se notar que a especificação DPWS proporciona características importantes para ser o núcleo de uma arquitetura de *middleware* que tem como objetivo interconectar diversos dispositivos em ambiente de computação ubíqua.

2.6 Trabalhos Relacionados

A literatura científica é rica em esforços de pesquisa que têm como objetivo prover interoperabilidade entre protocolos padrão de comunicação. Os trabalhos que serão analisados a seguir propõem soluções para permitir interconectividade entre tecnologias como Bluetooth e dispositivos RFID, Web Services e dispositivos RFID, UPnP e *Bluetooth*, UPnP e Jini ,UPnP e dispositivos RFID e também Jini e DPWS.

O trabalho de [Siegemund e Flörkemeier 2006], provê interconectividade entre as tecnologias *Bluetooth* e dispositivos RFID. O motivo da utilização da tecnologia *Bluetooth* decorre do crescente número de dispositivos que estão integrados com *Bluetooth*. Na abordagem dos autores, o *Bluetooth* funciona como um ponto de acesso móvel para as *tags* RFID ativas, permitindo que elas possam acessar serviços de infra-estrutura. Dessa forma, as *tags* ativas têm a possibilidade de realizar interpretações semânticas dos dados armazenados nas *tags* passivas por meio da estrutura desenvolvida no projeto.

No trabalho de [Prabakar, Kumar, Subrahmanya 2006] é proposto um *middleware* baseado em Serviços Web para gerenciamento de serviços. Os serviços que são gerenciados e disponibilizados são provenientes da tecnologia RFID, formando uma rede RFID. O trabalho foca em resolver falhas provenientes de equipamentos RFID e *softwares* associados, não se importando com falhas oriundas da infra-estrutura de rede. Os autores fazem uma análise do *middleware* Savant, que é responsável por gerenciar e monitorar atividades de leitores RFID. Esta análise aponta algumas falhas do *middleware* Savant, como perda de informação caso o *middleware* pare de funcionar, ou seja, as leituras de *tags* realizadas pelos RFID *Readers* podem ser perdidas. Para lidar

com esses problemas o trabalho propõe modificações na especificação Savant RFID, adicionando garantia na troca de mensagens com *WS-Reliability* e gerenciamento de recursos com *WS-Distributed Management*.

Outro trabalho propõe uma arquitetura para interconectar dispositivos das tecnologias *Bluetooth* e UPnP [Delphinanto, Lukkien, Koonen, et al 2007]. Nesta proposta foi projetada uma arquitetura na qual um *proxy* permite que serviços e dispositivos UPnP possam ser descobertos e executados por uma rede *Bluetooth* e vice-versa. Os autores desenvolveram um *proxy* que pudesse ser executado em um dispositivo com recursos computacionais limitados, dessa forma realizando um ponte entre uma rede *Bluetooth* e uma rede UPnP. Os dispositivos e serviços *Bluetooth* não possuem o mecanismo de anúncio como na arquitetura UPnP. Por isso, a arquitetura do projeto contempla o anúncio de serviços *Bluetooth* que entram na rede. O *proxy* também é responsável por converter padrões de descrições de serviços e dispositivos de *Bluetooth* para UPnP e de UPnP para *Bluetooth*.

Em [Allard, Chinta, Gundala, et al 2003] foi realizado um trabalho na qual é proposto um *framework* para permitir a interoperabilidade entre uma rede UPnP e uma rede Jini. Dessa forma, serviços UPnP podem descobrir e utilizar serviços Jini e vice-versa. O *framework*, além de realizar uma ponte entre padrões de tecnologias diferentes como Jini e UPnP, foi projetado para que novos serviços pudessem ser implementados reduzindo o tempo de desenvolvimento do canal de integração entre essas duas tecnologias. Dessa forma, o desenvolvimento do serviço teria seu foco na resolução do negócio. O *framework* provê componentes de descobrimento de serviços para Jini e UPnP, de registro de serviços encontrados e de conversão de padrão de descrição, além de invocação de serviços.

O trabalho proposto por [Hwang, Park, Chung 2008] tem como objetivo prover integração entre as tecnologias UPnP e RFID. A intenção do trabalho é rastrear a movimentação dos usuários em uma casa utilizando leitores RFID e ativar dispositivos que estejam em sua rota. Para isso, os usuários devem ter *tags* RFID associadas a eles. Os dispositivos dispostos pela casa lidam com a tecnologia UPnP. Sendo assim, foi necessário realizar uma estrutura de *middleware* entre os leitores RFID e os dispositivos UPnP. Essa estrutura permitiu guardar e mover uma sessão A/V (*Audio*/*Video*) do usuário de um dispositivo UPnP para outro a partir do momento que um leitor RFID detectasse a presença do usuário.

Pode-se notar que, independente da aplicação, o principal objetivo dos projetos apresentados é prover um canal de comunicação para troca de informações entre dispositivos de tecnologias distintas. Entretanto, as abordagens apresentadas tratam de duas tecnologias somente, não oferecendo uma solução abrangente para integração de dispositivos diversos de tecnologias heterogêneas em ambiente ubíquo.

Por outro lado, o projeto de [Raverdy, Riva, Chapelle, et al 2006] têm como objetivo permitir que dispositivos com diferentes tecnologias de rede possam ser interconectados. Os autores propõem o *middleware* MSDA (*Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments*) que permite compor e gerenciar diferentes padrões de comunicação como UPnP, *Bluetooth*, Jini e OSGi . A arquitetura projetada foi orientada por três princípios básicos: integração via *Middleware,* configuração dinâmica e MSDA como serviço. A integração via *Middleware* é responsável por estabelecer uma camada acima dos *middlewares* existentes criando uma interface única para que clientes possam acessar diferentes serviços e dispositivos de domínios diversos. Configuração dinâmica torna os

dispositivos MSDA independentes do ambiente de rede e permite que estes se comuniquem entre si para disseminar as informações sobre serviços remotos. MSDA como serviço é o princípio que define a forma como o *middleware* provê interfaces de descobrimento e acesso a outros serviços e dispositivos em ambiente de tecnologia específica, sendo disponibilizado ele mesmo como um serviço que pode ser utilizado pelos dispositivos da rede.

Outra proposta que abordada interoperabilidade para mais de uma tecnologia é apresentada por [Yim, Oh, Hwang e Lee 2007]. O *Web Services on Universal Networks* (WSUN), foi projetado para prover interoperabilidade entre tecnologias como DPWS e Jini. A estrutura da solução é composta por: US *Broker*, US *Registry* e os componentes adaptadores de tecnologias específicas como Jini *Adapter*. O principal componente do WSUN é o US *Broker*, que é responsável por realizar serviços de descobrimento, registrar serviços encontrados e gerenciar o registro desses serviços. O componente US *Registry* é responsável por armazenar informações sobre os dispositivos e serviços encontrados na rede e também o contexto (localização) desses dispositivos.

As soluções propostas em [Raverdy, Riva, Chapelle, et al 2006] e [Yim, Oh, Hwang e Lee 2007], apesar de fornecerem mecanismos que permitem que diferentes tecnologias possam ser integradas, requer uma infra-estrutura robusta para gerenciar dispositivos e serviços. Dessa forma, tanto o *middleware* MSDA quanto o WSUN não podem ser implantados em dispositivos com recursos de rede e computacionais limitados, como PDAs, *Smartphones* e *set-top boxes*. Em conseqüência disso, os pontos de acesso aos componentes do *middleware* ficam limitados a máquinas com poder de processamento satisfatório, não podendo aproveitar a gama de dispositivos existentes hoje como canal de comunicação para outras tecnologias.

2.7 Considerações Finais

Com relação às limitações apresentadas pelos projetos de pesquisa relatados, a proposta de um *middleware* de integração para interconectar todos os tipos de dispositivos, que empregam diversos tipos de tecnologias de comunicação, permanece uma oportunidade em aberto para pesquisa.

A seguir será apresentada uma infra-estrutura de middleware proposta para prover um canal de comunicação entre diversas tecnologias, podendo ser hospedado desde estações de trabalho a dispositivos com restrição de recursos.

3 *Device Service Bus*

O *Device Service Bus* (DSB) é uma infra-estrutura de *middleware* que objetiva prover integração de dispositivos heterogêneos em ambiente de computação ubíqua. O DSB é uma infra-estrutura de software baseada em uma plataforma portável e leve que pode ser executada desde em máquinas como estações de trabalho até em dispositivos com recursos computacionais limitados, como celulares e PDAs. Essa infra-estrutura é baseada no paradigma SOA, permitindo interação entre provedores e consumidores de serviço agindo como um *Service Broker*.

Figura 3.1. Visão Geral do *Device Service Bus*

O principal objetivo dessa infra-estrutura é criar um barramento de comunicação, mostrado na Figura 3.1, através do qual equipamentos com tecnologias de comunicação específicas, como RFID [Want 2006], Bluetooth [Chatschik 2001] e Sun SPOT [Sun 2004], possam cooperar uns com os outros, realizando serviços solicitados remotamente por seus pares. A interoperabilidade entre dispositivos é tornada possível pela adoção do padrão DPWS como tecnologia de base para construção do *middleware.* O DSB provê ainda um conjunto de componentes de integração para tecnologias de

comunicação específicas, como *Bluetooth* e RFID. Isso permite que dispositivos com recursos computacionais limitados ou não se integrem a outros dispositivos com suporte nativo para DPWS, mesmo que não possam executar toda a pilha de protocolos do DPWS.

A sessão seguinte deste capítulo apresenta os componentes de infra-estrutura do *middleware*, descreve seu funcionamento e analisa suas características em relação a outros trabalhos encontrados na literatura.

3.1 Componentes do *Middleware*

O *Middleware* proposto é composto por cinco tipos de componentes, que são mostrados na Figura 3.2.

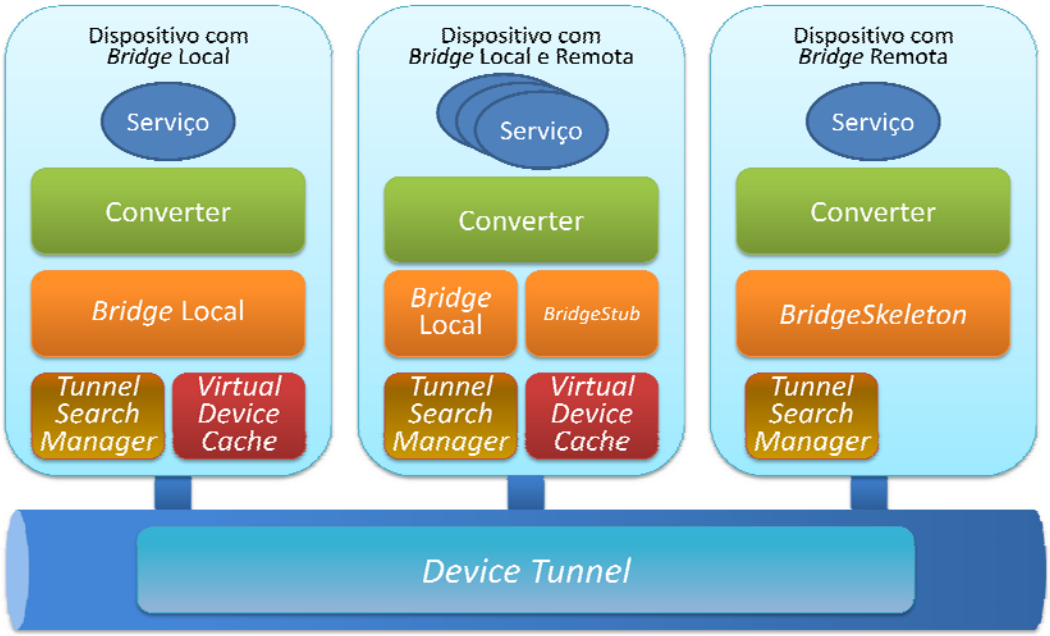

Figura 3.2. Componentes do *Device Service Bus*

O *Device Tunnel* é o componente capaz de descobrir seus pares, bem como outros dispositivos com suporte nativo para DPWS. O *Device Tunnel* foi construído como um *hosting service*, no qual é implementada a pilha de protocolos do DPWS. Todas as requisições, tanto de descobrimento via SOAP-sobre-UDP *multicast* e *unicast*, como de requisições RPC e baseadas em eventos via SOAP-sobre-HTTP, passam pelo *Device Tunnel*. Dessa forma, o *Device Tunnel* age como um despachante, distribuindo as requisições para os serviços associados a ele.

O *Virtual Device Cache* mantém uma lista de *Virtual Devices* conhecidos, que provêem interfaces DPWS para dispositivos que não implementam ou não suportam DPWS. Cada *Virtual Device* pode hospedar um ou mais *Virtual Services*, que representam serviços implementados por dispositivos não-DPWS. Além disso, um *Virtual Service* pode ter um ou mais *Virtual Action* e *Virtual Events*, que representam, respectivamente, as ações e eventos implementados por esses serviços.

A *Bridge* permite a interação entre dispositivos DPWS e *Virtual Devices*. Quando um *Virtual Device* é adicionado ou removido pela *Bridge*, mensagens de *Hello* e *Bye* são enviadas via *multicast*, conforme definido na especificação DPWS [Microsoft 2006]. Invocações de serviços enviadas para os *Virtual Devices* são encaminhadas para a *Bridge*, que as repassa para o *Converter* correspondente ao dispositivo. A *Bridge* pode ser tanto local (ou seja, pode executar no próprio dispositivo) como remota. A *Bridge* remota é constituída pelos serviços *BridgeStub* e *BridgeSkeleton*, sendo que o serviço *BridgeStub* fica hospedados no *Device Service Bus*, enquanto o serviço *BridgeSkeleton*, fica hospedado na *BridgeRemota*.Uma *Local Bridge* pode interagir com o *Device*

Tunnel local, enquanto a *Remote Bridge* precisa localizar um *Device Tunnel* através da rede.

Depois que um *Device Tunnel* é encontrado, a *Remote Bridge* precisa adicionar ao *Device Tunnel* todos os *Virtual Devices* gerenciados por ela, que são mantidos em sua *cache.* Em geral, opta-se por utilizar a *Remote Bridge* quando o dispositivo não tem capacidade suficiente para comportar todos os componentes do DSB. Conceitualmente, a *Remote Bridge* é similar ao *Discovery Proxy* definido pela especificação *WS-Discovery* [Microsoft 2005].

Um *Converter* gerencia um ou mais dispositivos que empregam a mesma tecnologia de rede. O *Converter* conhece detalhes de implementação dos dispositivos, sua interface e seus serviços. O *Converter* é responsável por extrair metadados dos dispositivos reais, fornecer a descrição dos seus serviços e por encaminhar invocações para tais serviços utilizando os mecanismos de comunicação suportados por cada dispositivo hospedeiro. Os *converters* mantêm uma *cache* de dispositivos convertidos que compreendem os metadados e descrições de dispositivos e serviços. A Figura 3.3 mostra um diagrama de classes que representa os objetos armazenados na *cache*.

O *Tunnel Search Manager* é responsável por realizar buscas por dispositivos e serviços requeridos pelos *Converters*. Essa busca pode ser realizada local e remotamente. No caso da busca local, o componente executa a requisição de busca na *Virtual Device Cache* do próprio DSB. A busca remota é realizada por meio do envio de mensagens *multicast* na rede (mensagem *Probe* do DPWS). Para cada dispositivo encontrado, seus metadados e a descrição dos serviços por ele fornecidos são repassados ao *Converter*, que nesse caso faz papel de cliente em busca de um provedor de serviços.

O *Converter* processa o retorno da requisição utilizando o mecanismo de comunicação suportado pelo dispositivo e a repassa para o dispositivo real. Para realizar a invocação, o *Converter* utiliza a *Remote Bridge* ou a *Local Bridge*.

O componente *Device Tunnel* expõe dispositivos e serviços encontrados pela *Bridge* e *Converters*. A *Bridge* manipula todos os *Converters* disponíveis e é responsável por conectar os *Virtual Devices* ao núcleo da arquitetura. O *Converter* lida com assuntos específicos da tecnologia para todos os dispositivos que empregam uma tecnologia em particular. Por essas razões, o DSB torna equipamentos que não implementam DPWS capazes de interagir com outros equipamentos através da sua infra-estrutura.

Os dispositivos conectados ao DSB têm seu próprio ciclo de vida, e podem ficar indisponíveis sem aviso prévio. Por isso, os *Converters* têm um mecanismo de *keep-alive* que mantém a rastreabilidade dos dispositivos ativos. Se um aviso de *time-out* ocorre para algum dispositivo, o *Converter* notifica a *Bridge*, que remove o dispositivo da *Virtual Device Cache*. O número de dispositivos que podem ser conectados ao DSB e o tempo de *time-out* do mecanismo de *keep-alive* podem ser configurados. Além disso, o mecasnimo de *keep-alive* pode ser desabilitado em ambientes com restrições de consumo de recursos computacionais e de rede.

3.2 Modelo de Classes do DSB

Os componentes *Converter, Bridges* e *Tunnel Search Manager* manipulam um conjunto de objetos que representam os dispositivos e serviços reais conectados à pilha

DSB. Esses objetos são armazenados em memória e estão diretamente associados a um *Converter*, pois é o *Converter* que realiza a manutenção desses objetos, inserindo e excluindo de sua *cache*, e os outros componentes realizam apenas consultas a esses objetos.

A classe *DeviceConverter*, ilustrada na Figura 3.3, representa o dispositivo convertido com seu metadado, o qual possui atributos como número serial, nome do dispositivo, nome do fabricante, modelo entre outros. Cada *DeviceConverter* possui uma lista de serviços que o dispositivo provê, sendo que cada serviço é modelada pela classe *ServiceConverter*. Cada *ServiceConverter* possui atributos como nome do serviço e uma lista de ações que possam ser executadas. As ações, por sua vez, são representadas pela classe *ActionConverter*. Esta pode ser dotada de parâmetros, que são representados pela classe *ParameterConverter*, que possui atributos como nome, direção (entrada ou saída), valor e tipo, representado pela classe *ParameterTypeConverter*.

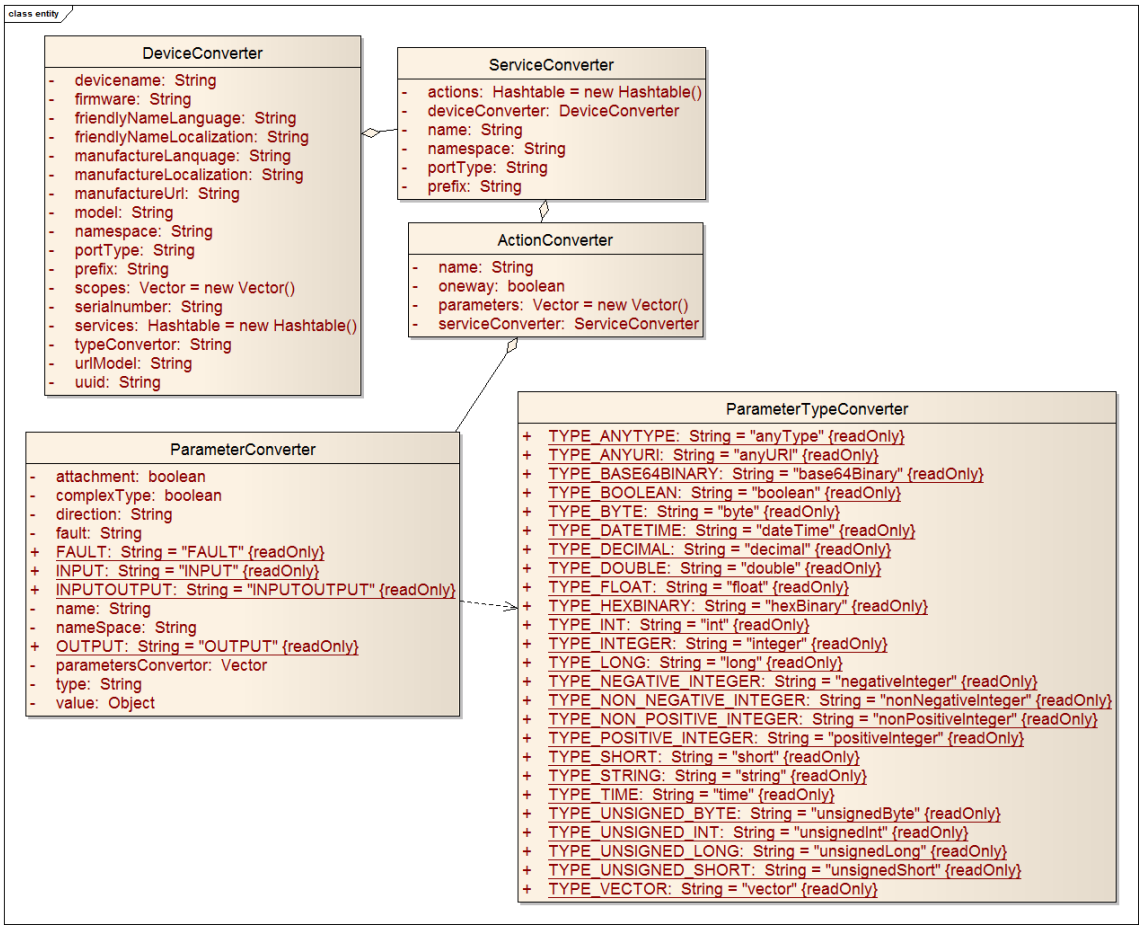

Figura 3.3. Objetos que representam os dispositivos e serviços convertidos

Exemplificando a representação desse modelo de classes, um *DeviceConverter* poderia ser uma impressora, dessa forma representando um dispositivo. Um *ServiceConverter* da impressora seria o serviço de impressão e ações desse serviço de impressão, *ActionConverter*, seria imprimir colorido e imprimir preto-e-branco. O *ParameterConverter* especificaria os parâmetros das ações do serviço de impressão.

Em fim, todas as invocações realizadas a dispositivos e seus serviços necessariamente manipulam o conjunto de objetos mostrados na Figura 3.3, que abstrai os dispositivos reais e unifica a forma como os componentes lidam com os equipamentos conectados ao DSB.

3.3 Dinâmica de Execução

O *Device Tunnel* lida com as requisições DPWS realizadas por um cliente em busca de dispositivos e/ou serviços. Como mostra a Figura 3.4, os seguintes passos são efetuados:

1. O cliente envia uma requisição *multicast* (*Probe*) na rede especificando o dispositivo a ser encontrado.
2. O Serviço *DeviceTunnel* recebe a requisição e verifica junto a *cache* de dispositivos virtuais se possui o dispositivo referente a requisição de busca.
3. Caso exista o metadado do dispositivo na *cache* de dispositivos virtuais, então o serviço *DeviceTunnel* envia uma mensagem *unicast* (*ProbeMatch*) como resposta ao cliente.
4. De posse do endereço do *HostingService*, o cliente envia uma mensagem (*Get*) para recuperar o metadado do dispositivo encontrado.
5. O *DeviceTunnel* busca na *cache* de dispositivos virtuais o metadado do dispositivo requerido.
6. O *DeviceTunnel* envia uma resposta (*GetResponse*) contendo o metadado do dispositivo ao cliente.
7. O cliente envia uma mensagem (*Get*) para recuperar a descrição do serviço hospedado pelo dispositivo em questão.
8. O *DeviceTunnel* busca na *cache* de dispositivos virtuais a descrição do serviço requerido.

9. O *DeviceTunnel* envia uma resposta (*GetResponse*) contendo a descrição do serviço ao cliente.

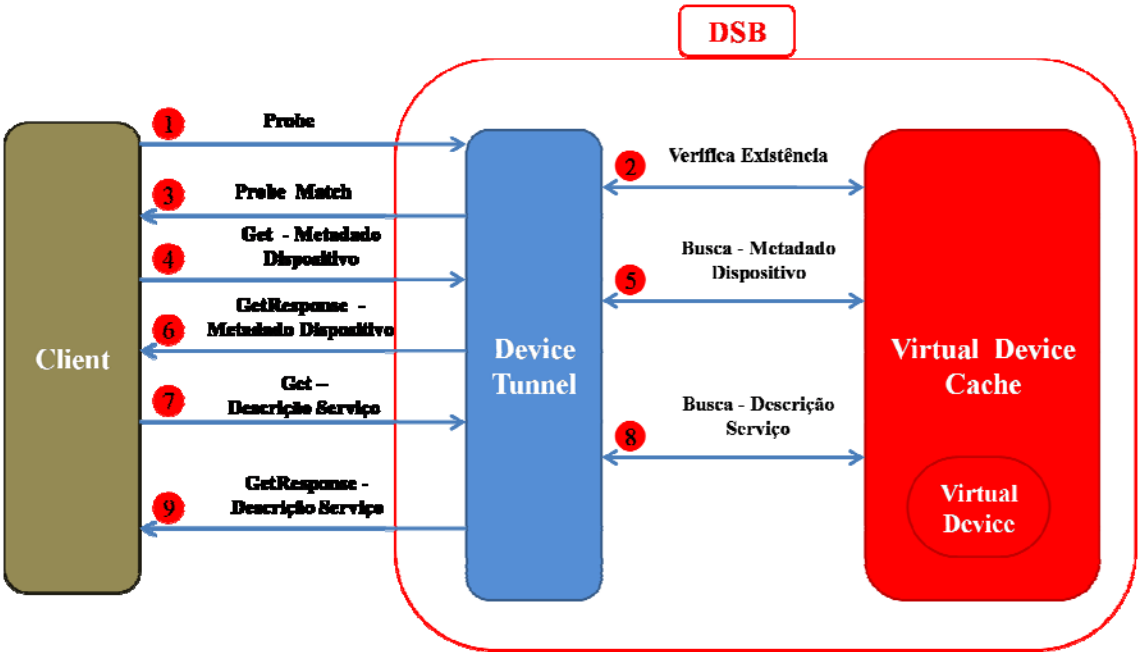

Figura 3.4. Descoberta de Dispositivos e Serviços

A Figura 3.5 demonstra o processo de invocação de um serviço, no qual os seguintes passos são realizados:

1. O cliente invoca o serviço para o DSB.
2. O *Device Tunnel* repassa a invocação para *Bridge*.
3. A *Bridge* repassa para o *Converter* a requisição.
4. Similarmente, o *Converter* identifica o dispositivo correspondente e lhe entrega a requisição, de modo que o mesmo possa a executar.
5. O *Converter* pode esperar pela execução de um serviço caso a requisição seja de *request-response*. Depois que o dispositivo executar a requisição, a resposta é transferida ao *Converter*.

6. O *Converter* retorna a resposta para *Bridge*.
7. A Bridge retorna a resposta ao *DeviceTunnel*.
8. O *DeviceTunnel* repassa a resposta ao cliente.

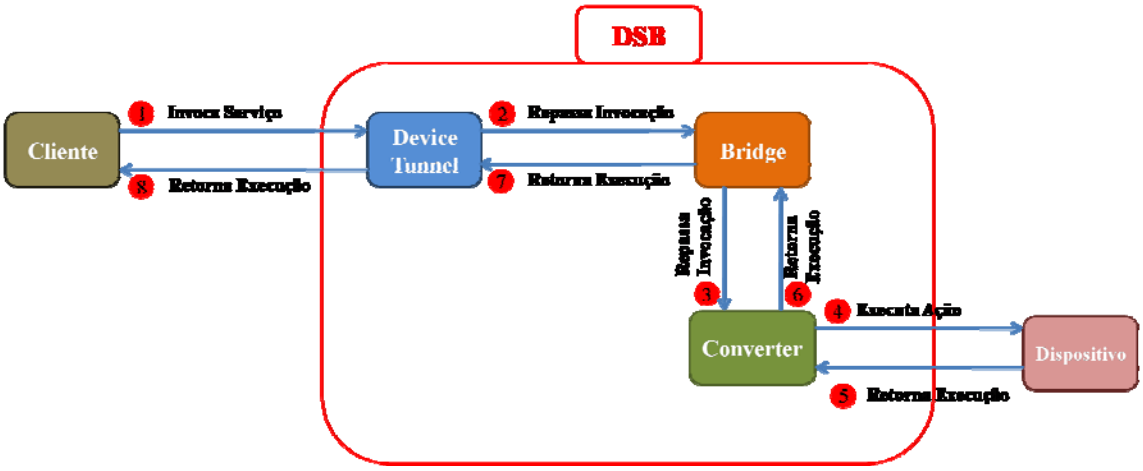

Figura 3.5. Invocação de Serviço

3.4 Descoberta e execução entre a *Remote Bridge* e o DSB

Com o objetivo de diminuir a quantidade de componentes a serem instalados em um dispositivo e prover conversores de tecnologias remotos, o componente *Remote Bridge* foi projetado, como um *Hosting Service* do DPWS, para atender a essas necessidades. A *Remote Bridge*, assim como a *LocalBridge*, consegue lidar com vários conversores de tecnologias, servindo a estes como canal entre o DSB e os dispositivos reais.

O DSB possui um serviço específico, o *BridgeStub*, que é responsável dentre outras funcionalidades, por lidar com os anúncios da *Remote Bridge* e realizar buscas por uma *Remote Bridge* . O *hosting service Remote Bridge* também possui um serviço específico, o *BridgeSkeleton*, que realiza buscas pelo DSB e também é capaz de lidar

com os anúncios do DSB. As mensagens de busca enviadas na rede pelos serviços *BridgeStub e BridgeSkeleton* são mensagens *Probe* (SOAP-sobre-UDP *multicast*) definidas pela especificação DPWS [Microsoft 2006]. Além disso, as mensagens que anunciam a entrada desses serviços na rede são as mensagens *Hello* (SOAP-sobre-UDP *multicast*). Caso a *Remote Bridge* ou o DSB lance o anúncio *Bye* (SOAP-sobre-UDP *multicast*) na rede, indicando a sua saída da rede, a *BridgeStub* ou *BridgeSkeleton*, que também lida com o anúncio *Bye*, descarta a referência armazenada da *Remote Bridge* ou do DSB. Caso o serviço *BridgeSkeleton* saia da rede, o serviço *BridgeStub* procura por um novo serviço *BridgeSkeleton*. Da mesma forma, caso um serviço *BridgeStub* saia da rede, o serviço *BridgeSkeleton* procura por um novo serviço *BridgeStub*. As figuras 3.6 e 3.7 mostram as interações de descobrimento e troca de endereços entre os serviços *BridgeStub* e *BridgeSkeleton*.

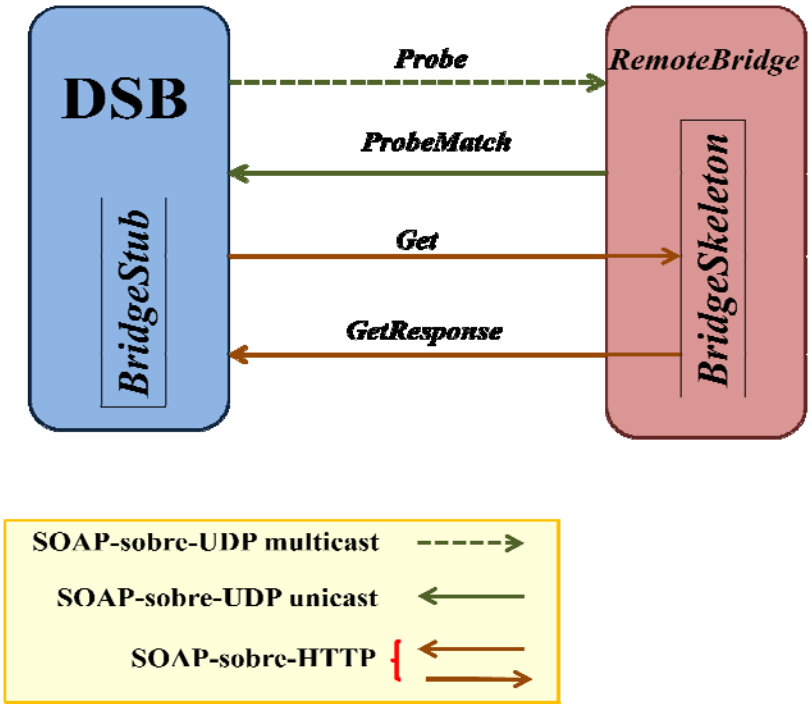

Figura 3.6. Descobrimento e troca de endereços entre a *BridgeStub* e *BridgeSkeleton*

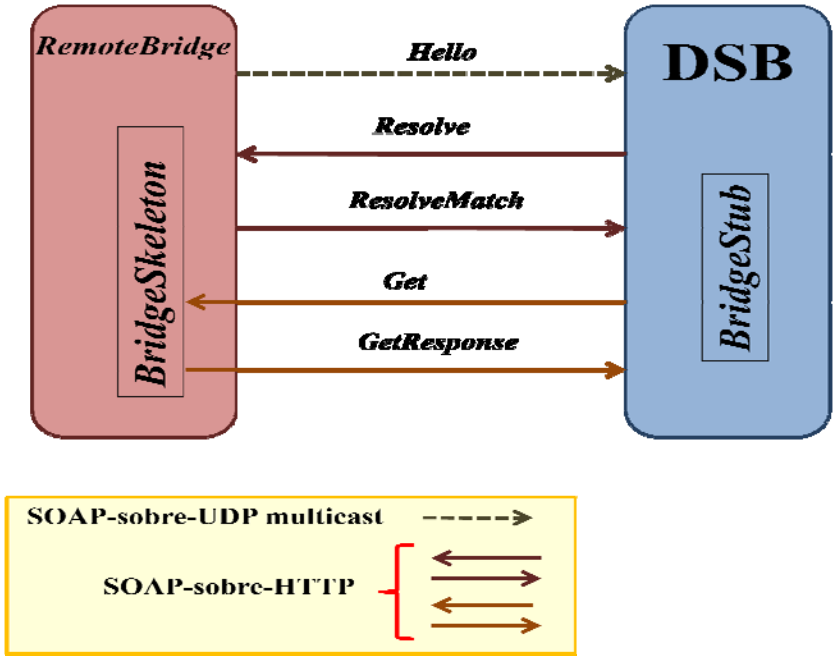

Figura 3.7. Anúncio e troca de endereços entre a *BridgeStub* e *BridgeSkeleton*

O descobrimento e troca de endereços entre o DSB e a *Remote Bridge* é a primeira etapa que consolida a extensão de funcionalidades do *Device Service Bus*. Como o DSB e a *Remote Bridge* são serviços independentes, cada um pode ser inicializado em tempos e em nodos diferentes. A *Remote Bridge*, juntamente com seus *Converters*, pode mapear e anexar dispositivos reais, alimentando a *cache* com objetos do tipo *DeviceConveters*, correspondentes aos dispositivos conecados ao DSB através do *Converter*. Caso a *Remote Bridge* tenha sido inicializada primeiro, ao descobrir e trocar endereços com um DSB, é realizada a transferência de todos os *DeviceConveters* da *Remote Bridge* para o DSB. O DSB, ao receber a requisição para adicionar dispositivos virtuais, cria um dispositivo virtual para cada *DeviceConverter* remoto, então é realizado o anúncio de cada dispositivo virtual por meio da mensagem *Hello* (SOAP-sobre-UDP *multicast*). À medida que novos dispositivos são anexados à *Remote Bridge*, o serviço *BridgeSkeleton* realiza a adição dos metadados dos dispositivos e

descrições de serviços no DSB por meio do serviço *BridgeStub*. Caso dispositivos sejam desconectados da *Remote Bridge*, o serviço *BridgeSkeleton* realiza a remoção dos metadados dos dispositivos e descrições de serviços no DSB por meio do serviço *BridgeStub*. Nesse caso, após remover o dispositivo virtual da *VirtualDeviceCache*, é enviada a mensagem de anúncio *Bye* (SOAP-sobre-UDP *multicast*) na rede.

As figuras 3.8 e 3.9 mostram a interface dos serviços *BridgeStub* e *BridgeSkeleton* respectivamente.

Após o DSB e a *Remote Bridge* terem trocado suas referências de rede, o canal de comunicação estendido está pronto para servir clientes com dispositivos tanto gerenciados pela *LocalBridge*, que fica junto à pilha do DSB, quanto pela *Remote Bridge*, que é remotamente referenciada pelo DSB.

```
<?xml version='1.0' encoding='UTF-8' ?>
<wsdl:definitions targetNamespace="http://www.ppgcc.inf.ufsc.br/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://www.ppgcc.inf.ufsc.br/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/">
  <wsdl:types>
    <xs:schema targetNamespace="http://www.ppgcc.inf.ufsc.br/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="VirtualDeviceList" type="tns:VirtualDevices" />
      <xs:complexType name="VirtualDevices">
        <xs:sequence>
          <xs:element name="RemoteVirtualDevice" minOccurs="0" maxOccurs="-1">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="DeviceName" type="xs:string" />
                <xs:element name="NameSpace" type="xs:string" />
                <xs:element name="PortType" type="xs:string" />
                <xs:element name="UUID" type="xs:string" />
                <xs:element name="Prefix" type="xs:string" />
                <xs:element name="Firmware" type="xs:string" />
                <xs:element name="Model" type="xs:string" />
                <xs:element name="UrlModel" type="xs:string" />
                <xs:element name="SerialNumber" type="xs:string" />
                <xs:element name="FriendlyNameLanguage" type="xs:string" />
                <xs:element name="FriendlyNameLocation" type="xs:string" />
                <xs:element name="Scope" type="xs:string" />
                <xs:element name="ManufactureLanguage" type="xs:string" />
                <xs:element name="ManufactureLocation" type="xs:string" />
                <xs:element name="ManufactureUrl" type="xs:string" />
                <xs:element name="TypeConverter" type="xs:string" />
                <xs:element name="VirtualService" minOccurs="0" maxOccurs="-1">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="PortType" type="xs:string" />
                      <xs:element name="NameSpace" type="xs:string" />
                      <xs:element name="Name" type="xs:string" />
                      <xs:element name="Prefix" type="xs:string" />
                      <xs:element name="VirtualActions" minOccurs="0" maxOccurs="-1">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="ActionName" type="xs:string" />
                            <xs:element name="Oneway" type="xs:boolean" />
                            <xs:element name="VirtualParameters" minOccurs="0" maxOccurs="-1">
                              <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="Name" type="xs:string" />
                                  <xs:element name="NameSpace" type="xs:string" />
                                  <xs:element name="Value" type="xs:string" />
                                  <xs:element name="Type" type="xs:string" />
                                  <xs:element name="Direction" type="xs:string" />
                                  <xs:element name="Fault" type="xs:string" />
                                </xs:sequence>
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="VirtualDeviceUUID" type="xs:string" />
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="AddRemoteVirtualDeviceinputMsg">
    <wsdl:part name="VirtualDeviceList" element="tns:VirtualDeviceList" />
  </wsdl:message>
  <wsdl:message name="RemoveRemoteVirtualDeviceinputMsg">
    <wsdl:part name="VirtualDeviceUUID" element="tns:VirtualDeviceUUID" />
  </wsdl:message>
  <wsdl:portType name="BridgeStub">
    <wsdl:operation name="AddRemoteVirtualDevice">
      <wsdl:input message="tns:AddRemoteVirtualDeviceinputMsg" wsa:Action="http://www.ppgcc.inf.ufsc.br/BridgeStub/AddRemoteVirtualDevice"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
    </wsdl:operation>
    <wsdl:operation name="RemoveRemoteVirtualDevice">
      <wsdl:input message="tns:RemoveRemoteVirtualDeviceinputMsg" wsa:Action="http://www.ppgcc.inf.ufsc.br/BridgeStub/RemoveRemoteVirtualDevice"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="BridgeStubBinding" type="tns:BridgeStub">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="AddRemoteVirtualDevice">
      <soap:operation soapAction="http://www.ppgcc.inf.ufsc.br/BridgeStub/AddRemoteVirtualDevice" />
      <wsdl:input>
        <soap:Body use="literal" />
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="RemoveRemoteVirtualDevice">
      <soap:operation soapAction="http://www.ppgcc.inf.ufsc.br/BridgeStub/RemoveRemoteVirtualDevice" />
      <wsdl:input>
        <soap:Body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="BridgeStubService">
    <wsdl:port binding="tns:BridgeStubBinding" name="BridgeStubPort">
      <soap:address location="http://192.168.1.101:58384/BridgeStub" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figura 3.8. WSDL do serviço *BridgeStub*

```
<?xml version='1.0' encoding='UTF-8' ?>
<wsdl:definitions targetNamespace="http://www.ppgcc.inf.ufsc.br/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://www.ppgcc.inf.ufsc.br/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/">
  <wsdl:types>
    <xs:schema targetNamespace="http://www.ppgcc.inf.ufsc.br/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="VirtualParameterInputList" type="tns:VirtualParametersInput" />
      <xs:complexType name="VirtualParametersInput">
        <xs:sequence>
          <xs:element name="VirtualParameterInput" minOccurs="0" maxOccurs="-1">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Name" type="xs:string" />
                <xs:element name="Value" type="xs:string" />
                <xs:element name="Type" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="VirtualServiceName" type="xs:string" />
      <xs:element name="VirtualDeviceUUID" type="xs:string" />
      <xs:element name="VirtualActionName" type="xs:string" />
      <xs:element name="VirtualParameterOutputList" type="tns:VirtualParametersOutput" />
      <xs:complexType name="VirtualParametersOutput">
        <xs:sequence>
          <xs:element name="VirtualParameterOutput" maxOccurs="-1">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Name" type="xs:string" />
                <xs:element name="Value" type="xs:string" />
                <xs:element name="Type" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="UUID" type="xs:string" />
      <xs:element name="VirtualParameterList" type="tns:VirtualParameters" />
      <xs:complexType name="VirtualParameters">
        <xs:sequence>
          <xs:element name="VirtualParameter" minOccurs="0" maxOccurs="-1">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Name" type="xs:string" />
                <xs:element name="Value" type="xs:string" />
                <xs:element name="Type" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="InvokeTwowayRemoteVirtualDeviceinputMsg">
    <wsdl:part name="VirtualParameterInputList" element="tns:VirtualParameterInputList" />
    <wsdl:part name="VirtualServiceName" element="tns:VirtualServiceName" />
    <wsdl:part name="VirtualDeviceUUID" element="tns:VirtualDeviceUUID" />
    <wsdl:part name="VirtualActionName" element="tns:VirtualActionName" />
  </wsdl:message>
  <wsdl:message name="InvokeTwowayRemoteVirtualDeviceoutputMsg">
    <wsdl:part name="VirtualParameterOutputList" element="tns:VirtualParameterOutputList" />
  </wsdl:message>
  <wsdl:message name="SearchBridgeStubinputMsg">
    <wsdl:part name="UUID" element="tns:UUID" />
  </wsdl:message>
  <wsdl:message name="InvokeOnewayRemoteVirtualDeviceinputMsg">
    <wsdl:part name="VirtualParameterList" element="tns:VirtualParameterList" />
    <wsdl:part name="VirtualServiceName" element="tns:VirtualServiceName" />
    <wsdl:part name="VirtualDeviceUUID" element="tns:VirtualDeviceUUID" />
    <wsdl:part name="VirtualActionName" element="tns:VirtualActionName" />
  </wsdl:message>
  <wsdl:portType name="BridgeSkeleton">
    <wsdl:operation name="InvokeTwowayRemoteVirtualDevice">
      <wsdl:input message="tns:InvokeTwowayRemoteVirtualDeviceinputMsg"
wsa:Action="http://www.ppgcc.inf.ufsc.br/BridgeSkeleton/InvokeTwowayRemoteVirtualDeviceRequest"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
      <wsdl:output message="tns:InvokeTwowayRemoteVirtualDeviceoutputMsg"
wsa:Action="http://www.ppgcc.inf.ufsc.br/BridgeSkeleton/InvokeTwowayRemoteVirtualDeviceResponse"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
    </wsdl:operation>
    <wsdl:operation name="SearchBridgeStub">
      <wsdl:input message="tns:SearchBridgeStubinputMsg" wsa:Action="http://www.ppgcc.inf.ufsc.br/BridgeSkeleton/SearchBridgeStub"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
    </wsdl:operation>
    <wsdl:operation name="InvokeOnewayRemoteVirtualDevice">
      <wsdl:input message="tns:InvokeOnewayRemoteVirtualDeviceinputMsg"
wsa:Action="http://www.ppgcc.inf.ufsc.br/BridgeSkeleton/InvokeOnewayRemoteVirtualDevice" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="BridgeSkeletonBinding" type="tns:BridgeSkeleton">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="InvokeTwowayRemoteVirtualDevice">
      <soap:operation soapAction="http://www.ppgcc.inf.ufsc.br/BridgeSkeleton/InvokeTwowayRemoteVirtualDeviceRequest" />
    </wsdl:operation>
    <wsdl:operation name="SearchBridgeStub">
      <soap:operation soapAction="http://www.ppgcc.inf.ufsc.br/BridgeSkeleton/SearchBridgeStub" />
      <wsdl:input>
        <soap:Body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="BridgeSkeletonService">
    <wsdl:port binding="tns:BridgeSkeletonBinding" name="BridgeSkeletonPort">
      <soap:address location="http://192.168.1.101:2105/BridgeSkeleton" />
    </wsdl:port>
```

Figura 3.9. WSDL do serviço *BridgeSkeleton*

A execução de serviços hospedados na *Remote Bridge* é realizada pelo canal criado entre o DSB e a *Remote Bridge* . Como os dispositivos e serviços anexados à *Remote Bridge* são expostos pelo DSB, os clientes realizam invocações através do DSB para chegar à *Remote Bridge* . Entretanto, as trocas de mensagens de descobrimento, requisições *Get*, *GetResponse*, *Resolve*, *ResolveMatch* são efetuadas entre o cliente e o DSB somente, visto que o DSB armazena o metadado dos dispositivos e a interface de seus serviços que estão anexados à *Remote Bridge* .

O serviço *BridgeSkeleton* possui duas ações: a *InvokeOnewayRemoteVirtualDevice* e *InvokeTwoWayRemoteVirtualDevice*, cujas interfaces estão ilustradas na figura 3.9. Estas são as ações utilizadas pelo serviço *BridgeStub* do DSB para repassar as invocações proveniente de clientes para a *BridgeSkeleton*. Ademais, o serviço *BridgeSkeleton* realiza a conversão do formato DPWS para o formato de objetos internos que está representado na Figura 3.3, que por sua vez despacha a requisição para o *Converter* apropriado. Este último invoca o dispositivo correspondente e pode ou não esperar por retorno. Caso seja uma invocação do tipo requisição-resposta, o *Converter* repassa o retorno da execução para a *BridgeSkeleton* e esta retorna para o DSB. Em seguida, o DSB devolve a execução ao cliente.

A ação *InvokeOnewayRemoteVirtualDevice* é utilizada para requisições que não esperam resposta, ou seja, invocações assíncronas. Já a ação *InvokeTwoWayRemoteVirtualDevice* é utilizada por requisições que esperam resposta, ou seja, invocações síncronas. A interface de entrada de dados para as duas interfaces é a mesma, sendo composta pelo identificador único universal (UUID – *Universal Unique Identifier*) do dispositivo alvo, o nome do serviço e da ação e uma lista de parâmetros da

ação requerida. A interface de saída de dados para a ação *InvokeTwoWayRemoteVirtualDevice* é uma lista de parâmetros definida pela ação do serviço.

As figuras 3.10 e 3.11 demonstram as trocas de mensagens na execução de requisições síncronas e assíncronas entre o serviço *BridgeStub* do DSB e o serviço *BridgeSkeleton* da *Remote Bridge* .

Figura 3.10. Execução da ação *InvokeOnewayRemoteVirtualDevice* da *BridgeSkeleton*

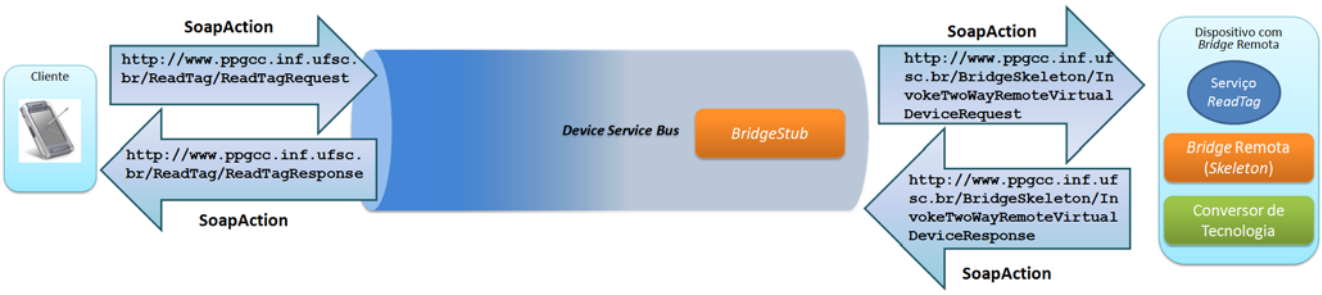

Figura 3.11. Execução da ação *InvokeTwoWayRemoteVirtualDevice* da *BridgeSkeleton*

O DSB provê aos dispositivos mecanismos de busca e execução de serviços providos por dispositivos conectados a ele. Esses mecanismos permitem que os dispositivos sejam ao mesmo tempo provedores e consumidores de serviços. Como descrito na seção 3.1, os C*onverters* podem utilizar o componente *Tunnel Search Manager* para realizar buscas por dispositivos e serviço. Essas buscas são realizadas

tanto na *cache* de dispositivos virtuais como na rede, por meio do envio de mensagem *multicast*. O *Converter* deve processar o retorno da requisição e realizar a conversão para a tecnologia suportada pelo dispositivo cliente. As invocações são realizadas por meio da *Bridge* Local e da *Bridge* Remota. Para dispositivos que estiverem anexados à *Bridge* Remota, as buscas do componente *Tunnel Search Manager* são realizadas tanto na *cache* de objetos do tipo *DeviceConverter* como por envio de mensagens *multicast* na rede. A figura 3.12 demonstra o funcionamento da busca de serviços realizada pelos dispositivos por meio do *Converter* no DSB, que ocorre da seguinte forma:

1. O dispositivo cliente efetua uma requisição de busca por outros dispositivos.
2. O *Converter*, que sabe lidar com a tecnologia do dispositivo cliente, converte a requisição para o formato do modelo de classe *DeviceConverter*, mostrado na Figura 3.3.
3. O *Converter* submete a requisição ao *Tunnel Search Manager*.
4. O componente *Tunnel Search Manager* realize uma busca local, na *cache* de dispositivos virtuais.
5. Caso exista o dispositivo requerido, é retornado o seu metadado ao *Tunnel Search Manager*.
6. O *Tunnel Search Manager* retorna a requisição para o *Converter*.
7. Caso o serviço não seja encontrado localmente, o *Tunnel Search Manager* realiza uma requisição do tipo *Probe* na rede em busca do dispositivo.
8. Caso o dispositivo exista, este é retornado ao *Tunnel Search Manager* que por sua vez retorna ao *Converter* o metadado do dispositivo.

9. De posse do metadado do dispositivo, o *Converter* converte o metadado para o formato da tecnologia suportada pelo dispositivo cliente.
10. O Converte retorna a o metadado do dispositivo encontrado ao dispositivo cliente.

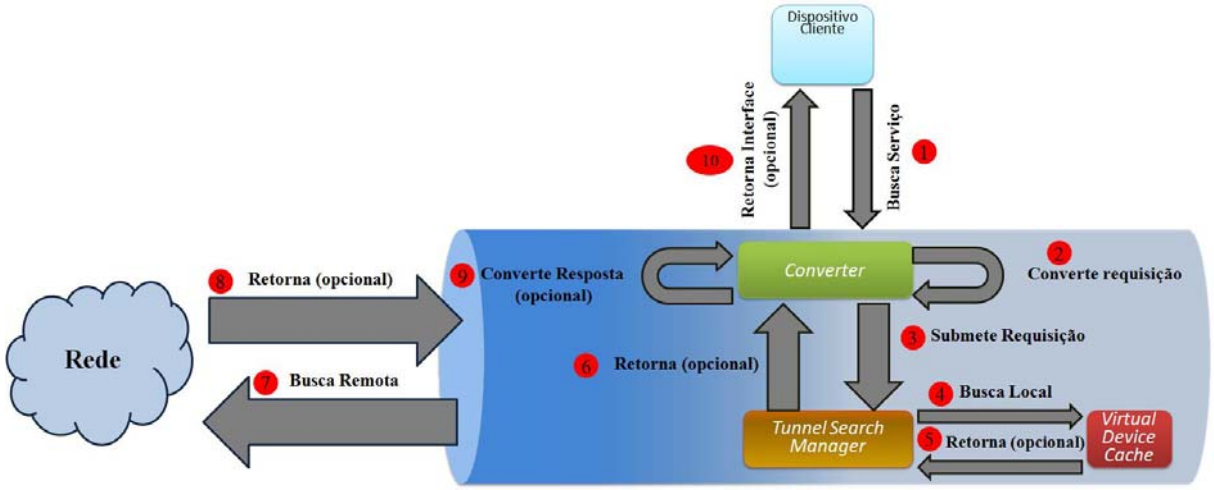

Figura 3.12. Busca por dispositivos e serviços a partir do dispositivo cliente

A Figura 3.13 ilustra a execução de serviços realizada pelos dispositivos por meio do *Converter* no DSB, e ocorre da seguinte forma:

1. O dispositivo cliente invoca o serviço ao *Converter*.
2. O *Converter* converte a requisição que está no formato da tecnologia do dispositivo cliente para o formato do modelo de classe *DeviceConverter*, mostrado na Figura 3.3.
3. O *Converter* submete a requisição para a *Bridge*.
4. A *Bridge* despacha a requisição para o *Converter* apropriado.
5. O *Converter* converte a requisição que está no formato do modelo de classe *DeviceConverter*, mostrado na Figura 3.3, para o formato da tecnologia do dispositivo provedor.
6. O *Converter* invoca o serviço.

7. O dispositivo pode retornar dados ao *Converter*.
8. O *Converter* converte a resposta do dispositivo provedor para o formato do modelo de classe *DeviceConverter*, mostrado na Figura 3.3.
9. O *Converter* retorna a resposta para o dispositivo provedor.
10. A *Bridge* retorna a resposta para o *Converter* que fez a requisição.
11. O *Converter* converte a resposta para o formato da tecnologia implementada pelo dispositivo cliente.
12. O *Converter* retorna os dados para o dispositivo cliente.

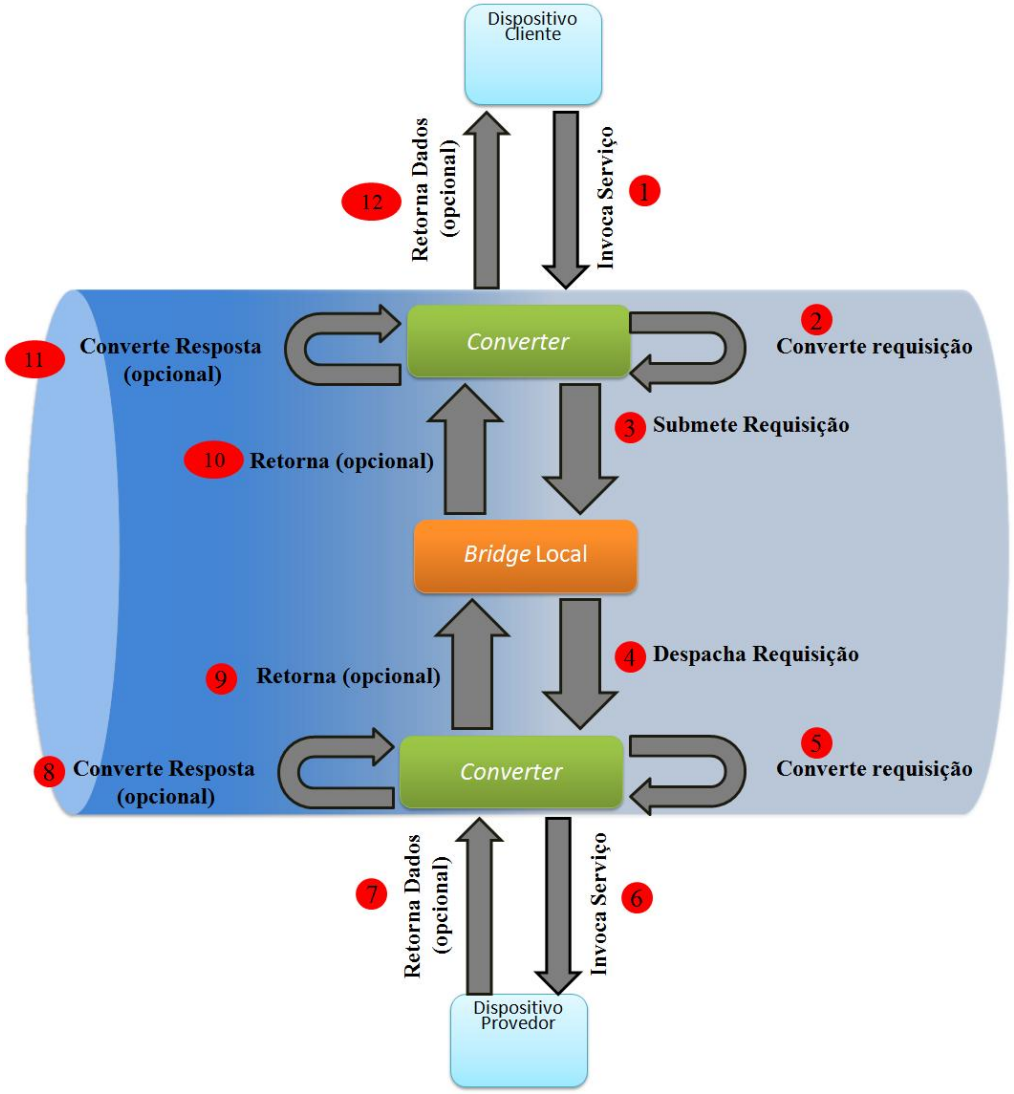

Figura 3.13. Execução de serviço invocado por dispositivo cliente

3.5 Análise

O DSB emprega uma abordagem para integração de dispositivos que é similar à estratégia adotada por outros *middlewares* de integração encontrados na literatura, como [Raverdy, Riva, Chapelle, et al 2006] e [Yim, Oh, Hwang e Lee 2007], o qual permite que dispositivos de diferentes tecnologias de rede interoperem. Entretanto, diferindo das outras propostas, a base do *middleware* DSB pode ser implantada em dispositivos com processadores 32 bits e que possuam no mínimo 2 MB de memória RAM, dessa forma podendo aproveitar dispositivos com poucos recursos computacionais, como PDAs, *Smartphones* e *Set-top boxes*. Além disso, o DSB pode ser disponibilizado em vários dispositivos, assim criando vários pontos de acesso com a finalidade de trocar informações através do DSB.

De acordo com [Jammes, Mensch e Smit 2005], o fato de a especificação DPWS empregar o *Discovery Proxy* permite que a área de cobertura das requisições seja ampliada. Além disso, pode-se transferir o processamento de descoberta para outra máquina, permitindo que dispositivos com recursos limitados sejam poupados. Na arquitetura do DSB, a *Bridge* pode ser executada remotamente, assim como o *Discovery Proxy*, beneficiando equipamentos com recursos computacionais limitados.

A base do projeto DSB é centrada no DPWS, que permite ao projeto ganhar com os benefícios do baixo acoplamento da arquitetura orientada a serviços, construída sobre protocolos como SOAP, HTTP e UDP *multicast* e *unicast*. A capacidade *plug and play*, que não está presente no serviço web comum, permite a descoberta dinâmica de serviços e simplifica o esforço necessário para expor serviços providos por dispositivos.

Essa característica é empregada pelo DSB para expor dispositivos e serviços que não suportam DPWS nativamente.

Esse capítulo apresentou a arquitetura do *Device Service Bus*, os seus componentes e o funcionamento das operações realizadas pela infra-estrutura. No capítulo seguinte serão apresentados os detalhes de implementação da arquitetura DSB, bem como, os testes de desempenho com quatro tipos de dispositivos, além disso, também serão apresentados dois cenários de uso para a solução DSB.

4 Implementação do *Device Service Bus*

A implementação do protótipo do *Device Service Bus* foi desenvolvida a partir do *framework* WS4D (*Web Service for Device*) [Zeeb, Bobek, Bohn, Prüter, Pohl, Krumm, Lück, Golatowski e Timmermann 2007], na qual o *Device Tunnel* foi construído e estendido para poder lidar com *Virtual Devices* e *Virtual Services*. Os componentes da pilha foram construídos em Java ME CDC 1.1.2.

Os *frameworks* WS4D e SOA4D foram analisados para o desenvolvimento do *Device Service Bus*. Estes *frameworks* são iniciativas de código aberto que disponibilizam a pilha do protocolo DPWS implementada. A seguir, estes *frameworks* de código aberto serão apresentados.

4.1 *Web Service for Device* – WS4D

O projeto *Web Service for Device*, resultado do projeto SIRENA [Bohn, Bobek e Golatowski 2006], é mantido pelas universidades alemãs Rostock e Dortmund e pela empresa MATERNA.

WS4D implementa a especificação DPWS e alguns *toolkits* foram construídos para prover uma implementação da especificação. Os *toolkits* são:

- WS4D – gSOAP : construído baseado no *toolkit* para serviços *web* gSOAP, é uma implementação realizada nas linguagens C/C++. Tem suporte para os sistemas operacionais Linux i386, Windows, Windows-cygwin e Linux embarcado. Foi projetado para ser executado em dispositivos com restrições de recursos computacionais.

- WS4D – Java *Multi-Edition*: foi construído na linguagem Java em três versões da *Java Virtual Machine*. Uma versão foi criada para Java SE, que foca máquinas sem restrição de recursos. Outras duas versões foram construídas para serem executadas em dispositivos com recursos computacionais limitados, cada versão nas máquinas virtuais Java ME CDC e CLDC.
- WS4D – Axis2: foi construído a partir do *framework* Apache Axis2, com intuito de suportar os serviços *web* construídos utilizando esse *framework.*

O grupo criador do WS4D mantém um grupo de discussão ativo, bem como provê atualização de versões dos três *toolkits*, disponibiliza documentação que descreve detalhadamente a arquitetura do projeto e seu modo de uso, e disponibiliza gratuitamente uma ferramenta para gerencia de dispositivos DPWS encontrados na rede. Essa ferramenta, o WS4D-Explorer, ilustrado na Figura 4.1, permite que sejam visualizados os dispositivos, serviços e ações, bem como suas descrições. Com essa ferramenta, também é possível invocar um serviço e visualizar o resultado, caso exista. Além disso, pode-se realizar busca por dispositivos e visualizar todas as trocas de mensagens na obtenção do metadado do dispositivo e das descrições de serviços.

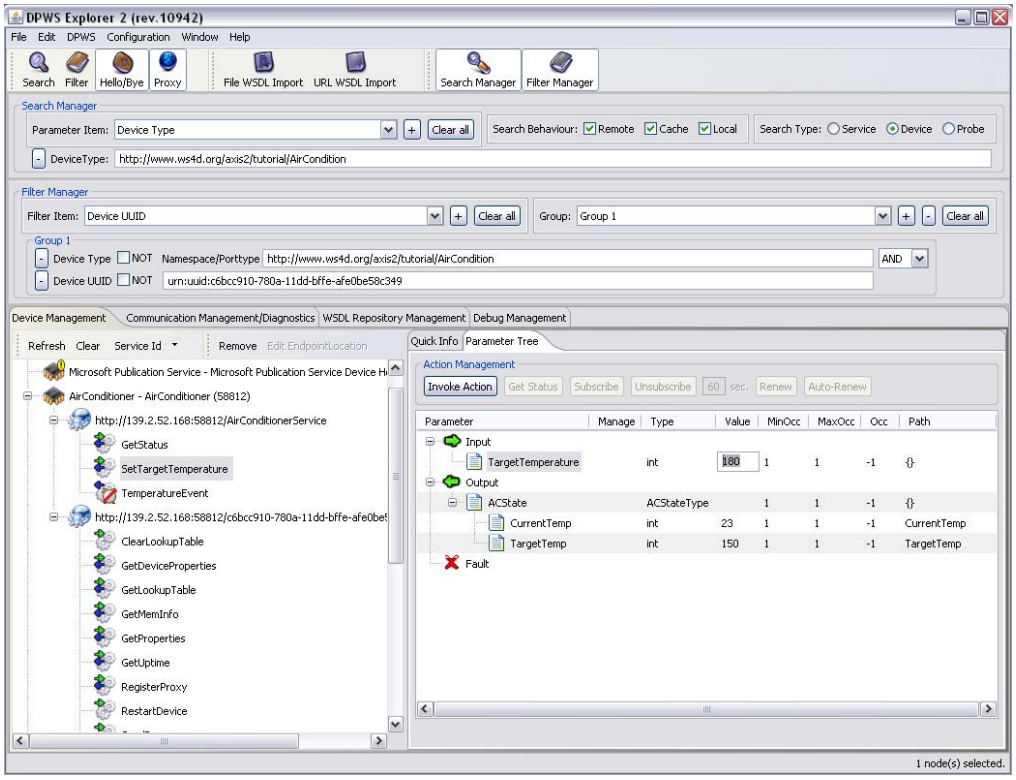

Figura 4.1. DPWS-Explorer

A arquitetura do WS4D-Java *Multi-Edition* foi projetada para ser modular, sendo possível estender cada módulo. Os principais módulos que formam o núcleo do *framework*, representados na Figura 4.2, são:

- org.ws4d.java.communication: contém interfaces responsáveis por padronizar a comunicação.
- org.ws4d.java.communication.soap: contém interfaces e implementações responsáveis por lidar com o protocolo SOAP.
- org.ws4d.java.communication.udp: contém interfaces e implementações responsáveis por lidar com o protocolo UDP.
- org.ws4d.java.communication.http: contém interfaces e implementações responsáveis por lidar com o protocolo HTTP.

- org.ws4d.java.service: contém interfaces responsáveis por padronizar os serviços do framework, como *hosting service* e *hosted service*.
- org.ws4d.java.service.discovery: contém interfaces e implementações responsáveis por efetuar o descobrimento de dispositivos e serviços.
- org.ws4d.java.service.remote: contém interfaces e implementações responsáveis por lidar com serviços remotos encontrados a partir de uma descoberta ou anúncio.

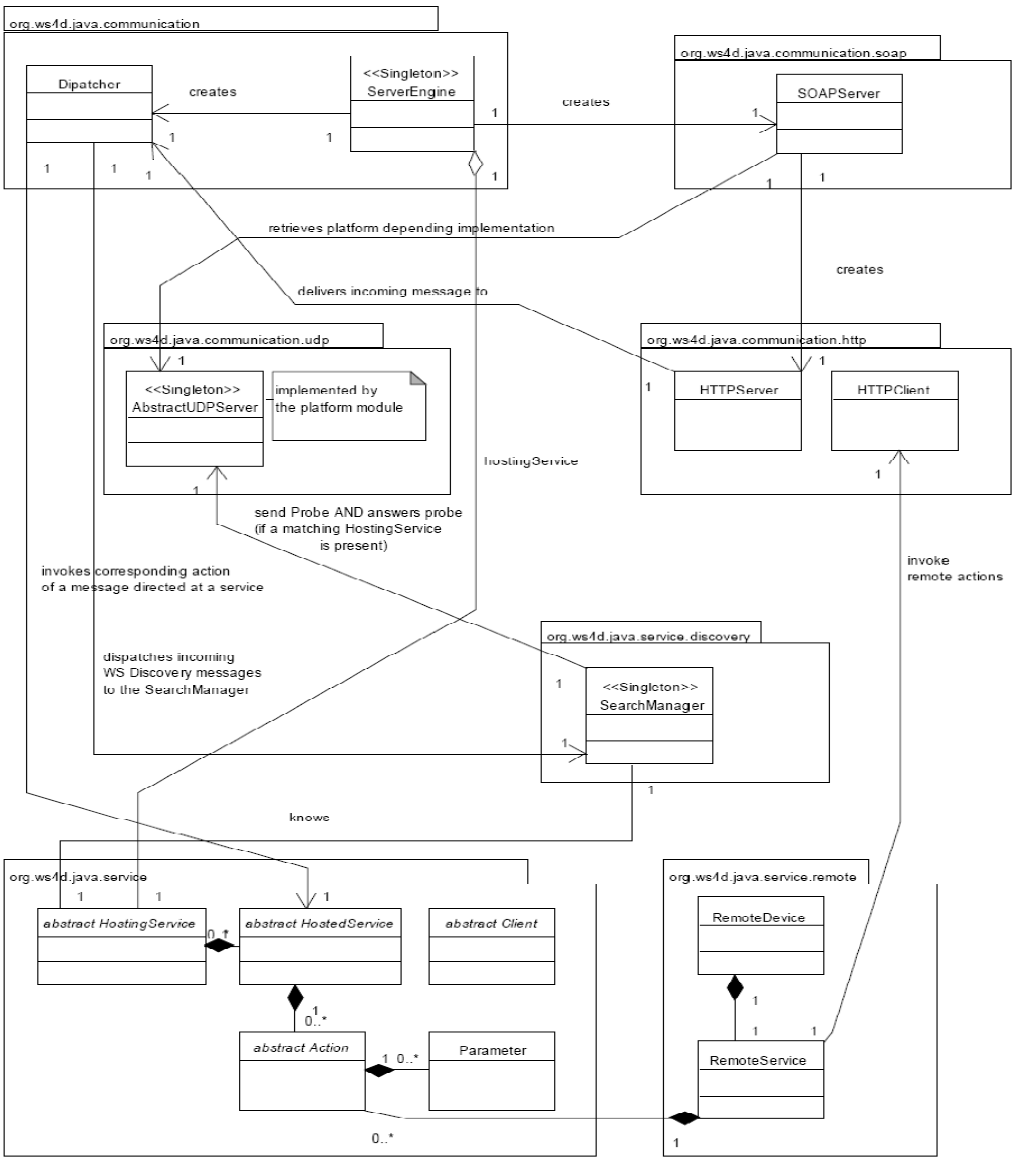

Figura 4.2. Principais Módulos do WS4D-Java *Multi-Edition*

Existem outras implementações da especificação DPWS como o SOA4D (*Service Oriented Arquitecture for Device*) que é realizada na linguagem C, DWPS4J (*Device Profile for Web Service for* Java) que é implementada na linguagem Java e também há uma implementação da Microsoft para .Net. Os *frameworks*, SOA4D e DPWS4J carecem de documentação que esclareça a arquitetura dos componentes, o modo como eles se relacionam e também uma documentação da API. Para que seja possível compreender e estender o *framework*, é preciso analisar o código fonte, o que pode ser feito empregando alguma ferramenta de modelagem e realizando uma engenharia reversa. A implementação em .Net não foi adotada por se tratar de código proprietário, sujeito a pagamento de licença.

Pelas características apresentadas nessa sessão optou-se pelo *framework* WS4D, por ser código aberto, ter uma ampla documentação, tanto da arquitetura, como da API, além exemplos que facilitam o desenvolvimento. Ademais, uma equipe atua ativamente no grupo de discussão possibilitando em pouco tempo sanar dúvidas e resolver erros que possam surgir com o uso da API.

4.2 Implementação dos Componentes do DSB

A presente seção apresenta alguns detalhes de implementação dos componentes do *Device Service Bus*.

Cada *Virtual Device* tem seu próprio metadado, que corresponde à descrição de um dispositivo real, i.e., número serial, número e nome do modelo, nome do fabricante e *friendlyname*, com o objetivo de atender a especificação DPWS. Um *Virtual Device* encapsula a descrição de *Virtual Services*, estabelecendo uma associação de um-para-muitos. O *Device Tunnel* expõe como serviço web todos os V*irtual Devices* e V*irtual*

Services, associando um endereço único para cada um deles. Este endereço é usado pelo *Device Tunnel* para demultiplexar requisições e informar à *Bridge* qual *Virtual Device* e *Virtual Service* foram invocados.

A *Bridge* mantém em sua *cache* uma lista de *Converters*, e cada *Converter* mantém um ponto de conexão para os dispositivos que empregam uma tecnologia específica. Além disso, a *Bridge* transforma o formato DPWS para o modelo de domínio conhecido pelos *Converters*, que são os *DeviceConverter*, *ServiceConverter*, *ActionConverter* e *ParameterConverter*, como mostrado na Figura 3.1 do capítulo 3, e também realiza a transformação inversa.

A *Bridge* Local foi implementada como o padrão de projeto *singleton*, dessa forma, sua instância é inicializada somente uma vez no momento de inicialização do DSB. A Figura 4.3 ilustra o diagrama de classe que envolve a *Bridge* local com as classes *Converter*, *BridgeStub* e as classes internas ao *BridgeStub*.

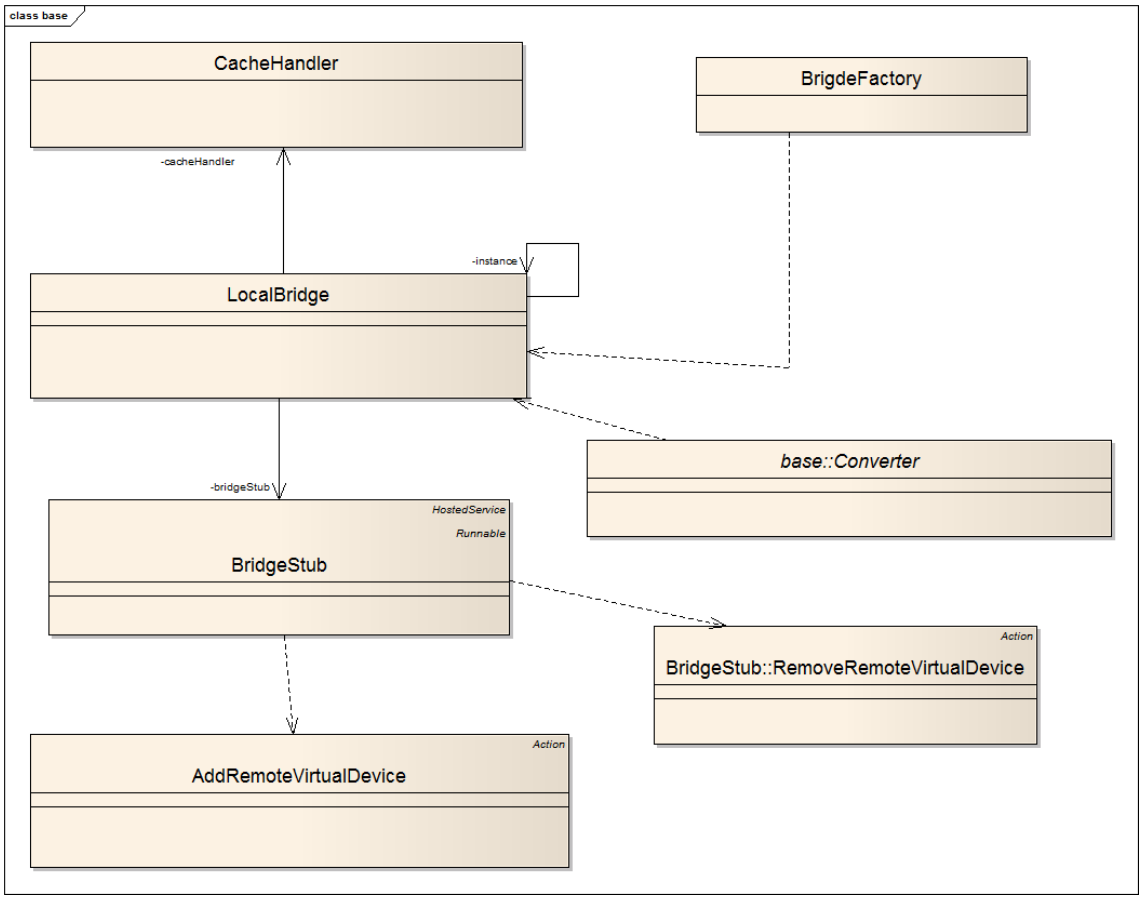

Figura 4.3. Diagrama de Classes da *Bridge* Local

As interações entre o DSB e a *Remote Bridge* são realizadas por meio dos serviços *BridgeStub* e *BridgeSkeleton*. Como é a *Remote Bridge* que provê os dispositivos reais ao DSB, o conceito *Skeleton* ficou no serviço hospedeiro *Remote Bridge* e o *Stub* no DSB. Todas as mensagens trocadas pelos dois serviços são no formato DPWS, desde o descobrimento e troca de descrições de serviço até a invocação da ação do serviço. O diagrama de classes envolvendo a *BridgeSkeleton* da *Remote Bridge* é mostrado na Figura 4.4.

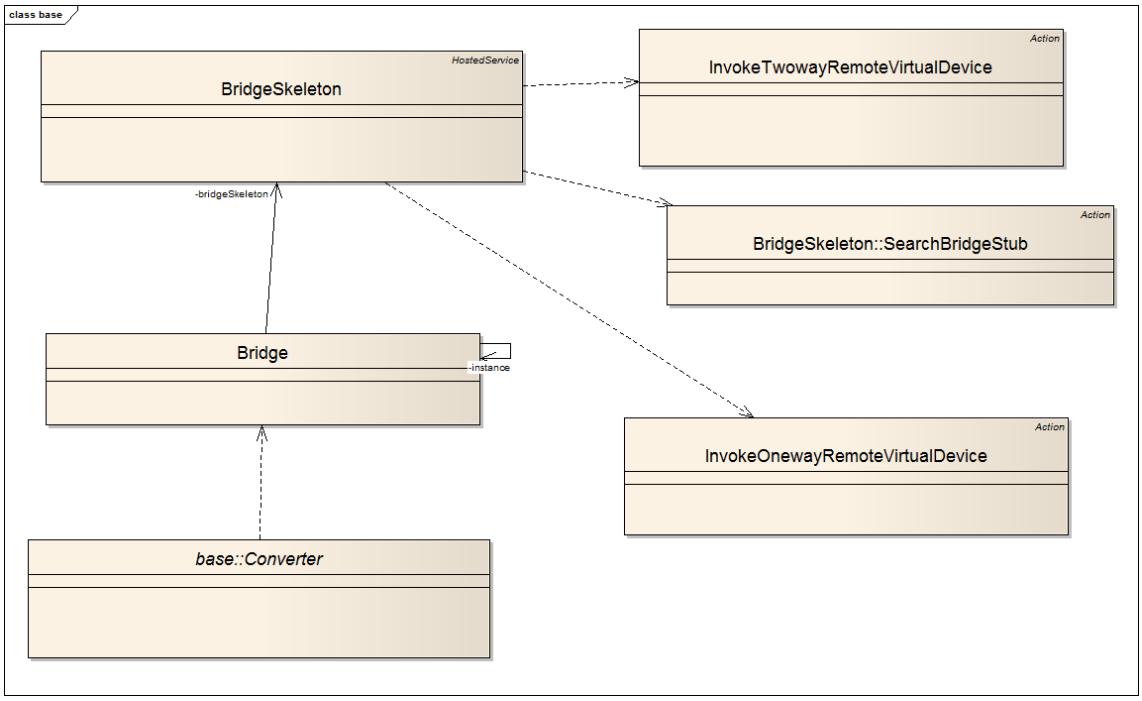

Figura 4.4. Diagrama de classes da BridgeSkeleton.

Na descrição WSDL do serviço *BridgeStub*, a ação de adicionar dispositivo, aceita como parâmetro uma lista de dispositivos para serem adicionados. Cada dispositivo deve ser associado a uma lista de serviços, e cada um desses deve ter uma lista de ações, com cada ação possuindo uma lista de parâmetros. Já a ação de remoção de dispositivos aceita o identificador do dispositivo UUID, pois é preciso somente do identificador do dispositivo para removê-lo da *cache* de dispositivos virtuais.

Os componentes *Converters* foram implementados herdando a abstração *Converter*, que sabe lidar com o domínio dos *DeviceConverter*, realizar chamadas às *Bridges*, tanto remota como localmente, e lidar com o componente *Tunnel Search Manager* montando requisições de busca por dispositivos e serviços, agindo como cliente. Cada *Converter* sabe lidar com uma tecnologia específica, dessa forma, foram

implementados *Converters* para lidar com as tecnologias Bluetooth, RFID e com sensores Sun SPOT.

A classe abstrata *Converter* apresentada na Figura 4.5 implementa a interface *ITunnelSearchCallback*, que é responsável por prover dois métodos de *callback* para realização de buscas por dispositivos e serviços efetuados pela classe *TunnelSearchManager*. Esse métodos de *callback* são responsáveis por receber a interface de descobrimento *TunnelSearchResult* que encapsula o dispositivo descoberto, *DeviceConverter*. Cada método de *callback* é invocado pela classe *TunnelSearchManager* de acordo com o alvo da busca realizada pelo *Converter* que pode ser um dispositivo ou dispositivo e serviço.

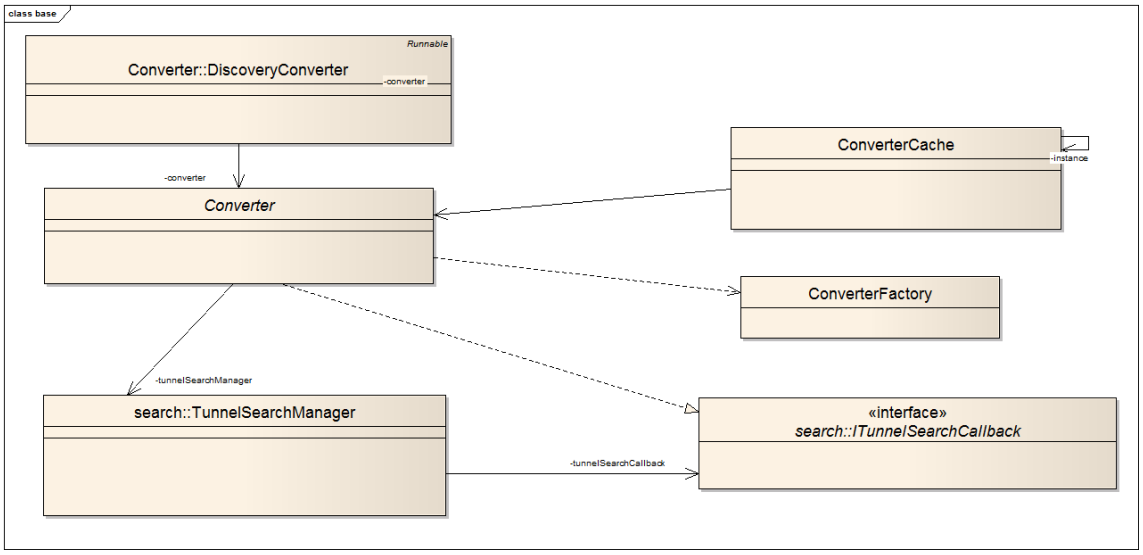

Figura 4.5. Diagrama de Classe do componente *Converter*

A classe Converter possui métodos que são utilizados para manipulação do domínio na *chache* de *DeviceConverter*, representada pela classe *ConverterCache* demonstrada na Figura 4.5, invocação de serviços anexados ao C*onverter* específico e

também execução de serviços provenientes de buscas realizadas pelo *TunnelSearchManager*.

A classes e interfaces envolvendo o componente *Tunnel Search Manager* são apresentadas na Figura 4.6. A classe *TunneSearchManager* utiliza a *Thread SearchListener* para realizar as buscas remotas, e as requisições são manipuladas utilizando a classe *TunnelSearchRequest* para encapsular o formato de pedidos de busca por dispositivos e serviços. Já a classe *TunnelSearchResult* é utilizada para encapsular o retorno do descobrimento, que contém o metadado do dispositivo e a descrição do serviço requerido.

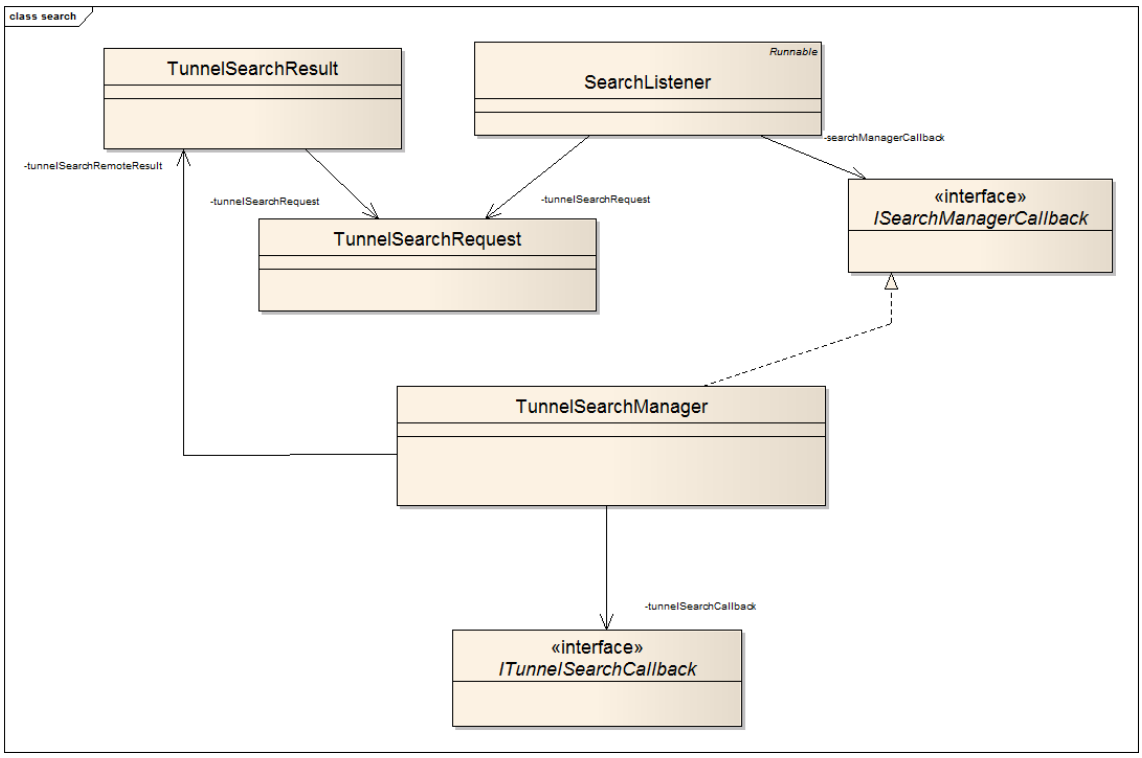

Figura 4.6. Diagrama de classes do componente *TunnelSearchManager*

A implementação de cada *converter* será apresentada a seguir, descrevendo os detalhes de tecnologia utilizada.

4.3 *Converter* RFID

Para lidar com a tecnologia *Radio Frequency Identification* foram implementados dois *Converters* – um para o leitor Mercury M5e, e outro para o leitor Mercury M5, ambos fabricados pela empresa *ThingMagic*. Os leitores M5e e M5 têm capacidades diferentes de leitura e características diferentes de fabricação.

O M5e possui um *hardware* limitado e permite que no máximo duas antenas sejam conectadas a ele, permitindo leituras a pouco mais 1 metro. A interface de comunicação oferecida pelo equipamento é a conexão serial. Comandos hexadecimais são usados para realizar operações de leitura, escrita e consultar dados como número de série do equipamento. As leituras são realizadas e armazenadas em uma memória *flash*, da qual é possível recuperar dados como o EPC (*Eletronic Product Code*) da etiqueta.

As características do leitor M5e são:

- Processador Atmel AT91SAM7S-256
- Memória *Flash* 256 Kb
- Memória RAM 64 Kb
- Arquitetura de Rádio Freqüência ASIC Intel R1000
- Protocolos suportados GEN2, ISO 18000-6C
- Conexão Serial

O *ConverterM5e* foi construído para lidar especificamente com o leitor M5e. Uma biblioteca foi desenvolvida para manipular os comandos hexadecimais como conectar e reiniciar o dispositivo, retornar dados de metadado e realizar leituras e escritas em etiquetas.

Já o leitor M5 possui um *hardware* mais elaborado, com maior capacidade de processador e de memória *flash*, além de oferecer entradas para quatro antenas, podendo

realizar leituras de até oito metros de distância das etiquetas. O M5 também provê conexão TCP/IP e aceita comando RQL (*Reader Query Language*), uma metalinguagem que permite facilitar as implementações dos comandos.

As características do M5 são:

- Processador Intel IXP420 266 MHz.
- Memória de 128 Mb DRAM.
- Memória *flash* de 32 Mb.
- Conexão Ethernet 10/100 Mb.
- Sistema operacional MercuryOS (Linux *kernel* versão 2.4.x).
- Suporta protocolos de rede: TCP/IP, UDP/IP, HTTPS, HTTP, SSH/SSL.
- Suporta servidor web CGI.
- Suporta *Reader Query Language* (RQL).
- Suporta *Dynamic Host Configuration Protocol* (DHCP).
- Suporta os protocolos EPC Class 0, EPC Class 1, Gen1 and Gen2, ISO 18000-6B/Ucode 1.19 e Rewriteable Class 0+.
- Oito entradas para antennas.

Para o leitor M5 foi desenvolvido o *ConverterM5*, assim uma biblioteca também foi construída com o intuito de lidar com requisições TCP/IP e manipular comandos RQL, para selecionar, filtrar e escrever dados nas etiquetas.

Como todos os *Converters*, tanto o *ConverterM5e* quanto o *ConverterM5* também convertem requisições, que nesse caso são de leitura e escrita de etiquetas, provenientes das *Bridges*, para comandos hexadecimais e comandos RQL. Além disso, o retorno também é transformado para o domínio do *DeviceConverter*. As etiquetas que

foram usadas para os experimentos foram etiquetas passivas com 96 bits de capacidade de armazenamento.

As classes que envolvem os *Converter* para os leitores RFID Mercury M5e e Mercury M5 são mostradas na Figura 4.7. A classe *M5eReaderConverter* encapsula uma biblioteca *M5eHandler* para lidar especificamente com o leitor M5e, enquanto a classe *M5ReaderConverter* encapsula a biblioteca *M5Handler* para lidar especificamente com o leitor M5.

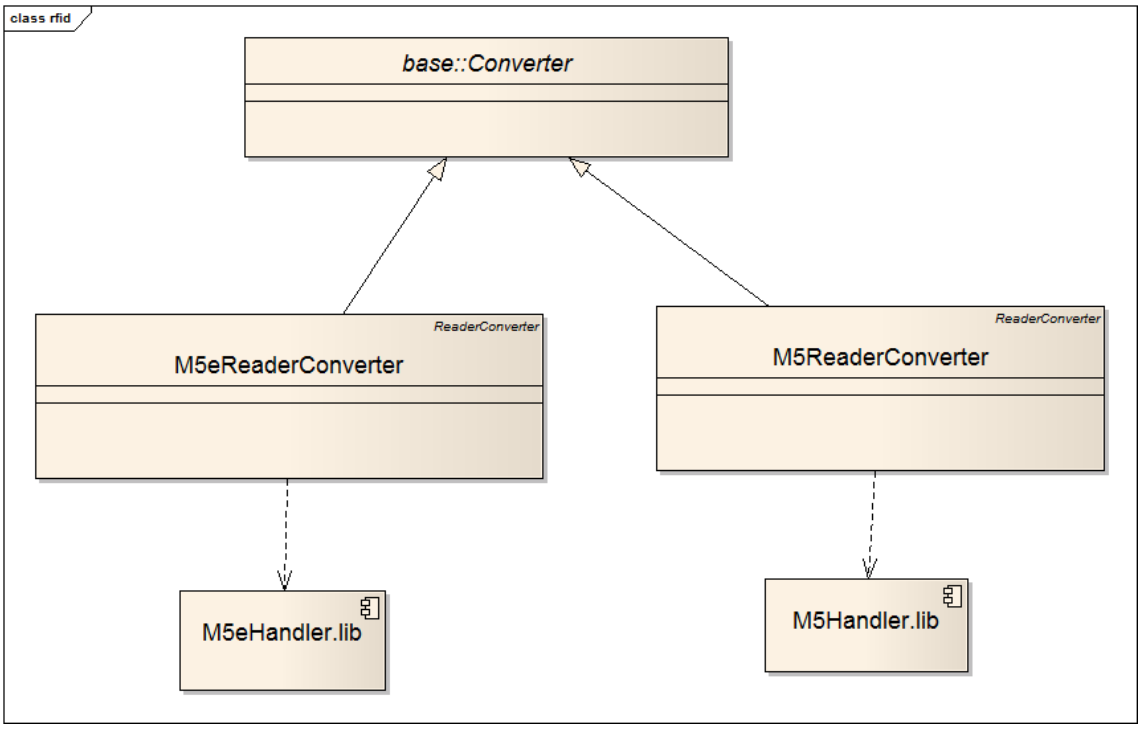

Figura 4.7. Diagrama de Classes dos Converter para tecnologias RFID com suas respectivas bibliotecas.

4.4 *Converter* Sun SPOT

O sensor Sun SPOT, fabricado pela empresa Sun Microsystems, também foi estudado a fim de poder ser construído um conversor para esse tipo de dispositivo. Dessa forma, os dispositivos sensores e seus serviços de leitura de temperatura e intensidade de luz puderam ser expostos.

O *firmware* do Sun SPOT é uma *virtual machine* (VM) desenvolvida especialmente para esse tipo de dispositivo, chamada de Squark. A VM da plataforma Squark suporta implementações em Java com API da plataforma Java ME (Java *Micro Edition*) para desenvolvimento de aplicações embarcadas.

As características do sensor Sun SPOT são:

- Padrão de comunicação IEEE 802.15.4, a 2.4GHz.
- Processador ARM modelo 920T de 32 bits com 180 MHz.
- Memória de 512K e 4M de memória flash.
- Medidor de aceleração de 3 eixos, 2G/6G.
- Sensor de Luz.
- Sensor de Temperatura.
- Seis entradas analógicas e 8 LEDs.
- Bateria recarregável e gerenciada pela VM Squark.
- Máquina Virtual Squark que executa diretamente no processador.

Para desenvolvimento de aplicações embarcadas com o sensor Sun SPOT, a Sun Microsystems disponibiliza um kit de desenvolvimento constituído por dois sensores Sun SPOT, uma estação base Sun SPOT, também chamada de *host* e um cabo USB.

A configuração de desenvolvimento constitui da estação de base conectada a um computador via cabo USB e os sensores Sun SPOT comunicando entre si e com a estação base via ondas de rádio. É possível construir aplicativos Java com a VM Squark e fazer uma implantação nos sensores via cabo USB.

Em virtude de poder disponibilizar esse dispositivo e seus serviços de leitura de intensidade de luz e temperatura, foi desenvolvido o converter *SunSpotConverter*. Para que uma aplicação externa se comunique com o sensor Sun SPOT, é necessário construir uma aplicação do tipo *host* e anexar algumas bibliotecas da VM Squark. Dessa forma, a aplicação *host* utiliza a estação base para se comunicar e trocas dados com os sensores. Também foi desenvolvido um aplicativo para os sensores, no intuito de monitor a temperatura e intensidade de luz e ainda se comunicar com a estação-base.

Foram construídos duas bibliotecas para dar suporte aos dispositivos Sun SPOT. A biblioteca *SunSpotHandler* está diretamente acoplada ao *converter SunSpotConverter* e possui um servidor TCP/IP para se comunicar com a segunda biblioteca *SunSpotHost*, que também possui um servidor TCP/IP. A biblioteca *SunSpotHost* é responsável por realizar a comunicação e troca de informações com os sensores Sun SPOT por meio da estação base, por isso é necessário que esta biblioteca execute a máquina virtual Squark. Na comunicação entre as bibliotecas *SunSpotHandler* e *SunSpotHost* são trafegados objetos serializados Java.

A intenção da separação dessas duas bibliotecas e a comunicação via TCP/IP,foi decidida para isolar o *converter SunSpotConverter* da Máquina Virtual Squark. Dessa forma, o dispositivo que for hospedar a pilha DSB com o converter *SunSpotConverter* não precisa executar a máquina virtual Squark. Assim, é possível ter parte do *converter SunSpotConverter* em outro dispositivo.

O componente *SunSpotConverter* também realiza a conversão dos metadados e descrição de serviços dos sensores encontrados para o domínio *DeviceConverter* e os adiciona por meio das *Bridges* na *cache* de dispositivos virtuais. Ademais, o *SunSpotConverter* converte as requisições repassadas pelas *Bridges* para requisições no padrão de comunicação 802.15.4, no formato da plataforma Squark. A Figura 4.8 mostra o diagrama de classes do componente *SunSpotConverter* e as suas bibliotecas.

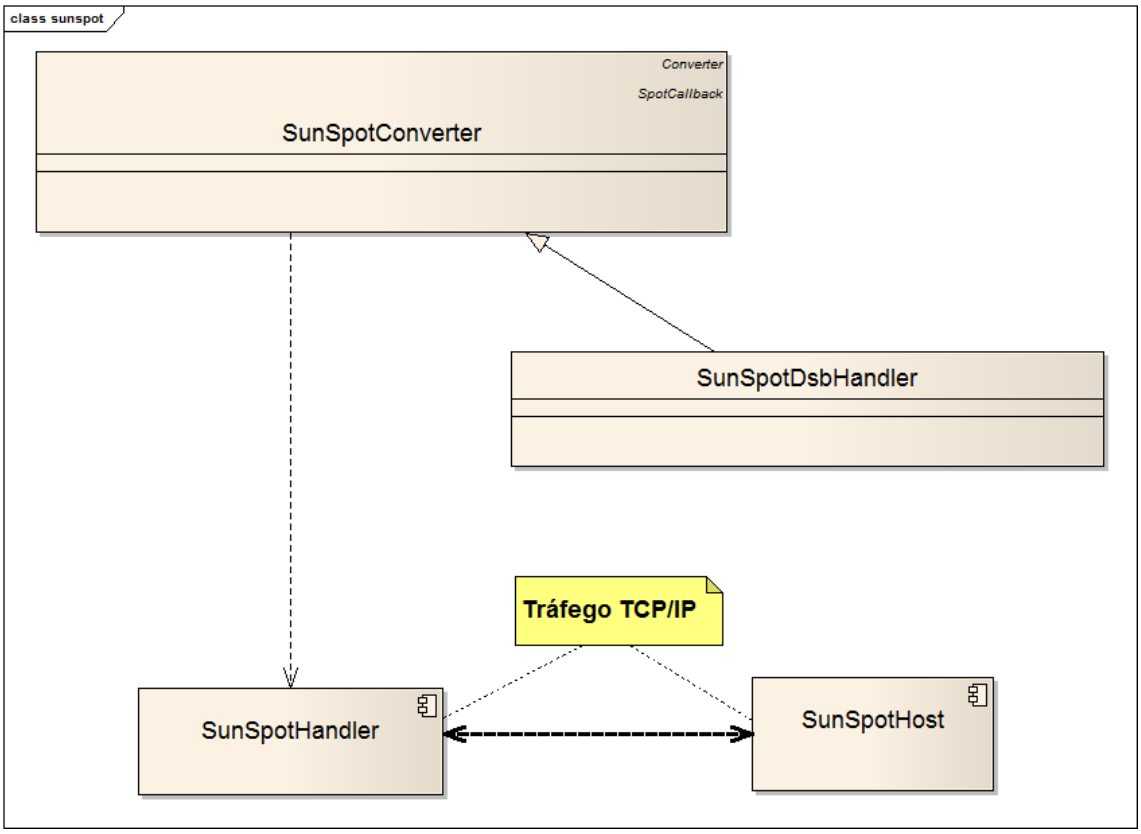

Figura 4.8. Diagrama de Classes do componente *SunSpotConverter*

4.5 *Converter Bluetooth*

O Conversor para o padrão de comunicação *Bluetooth* também foi realizado como parte do trabalho de conclusão do aluno Fábio Kreush, do curso de Sistemas de

Informação da Universidade Federal de Santa Catarina (UFSC), sob supervisão do autor da presente dissertação e de seu orientador. No momento da elaboração dessa dissertação, a monografia do aluno encontrava-se em elaboração, sendo seu título provisório “Integração de *Web Services* com dispositivos móveis através de *Bluetooth*”. A descrição a seguir é baseada em resultados preliminares obtidos pelo aluno.

Seguindo a arquitetura do DSB, o *BluetoothConverter* herdou a abstração *Converter*, dessa forma, permitindo adquirir todas as funcionalidades do conversor e o conectando com os outros componentes do DSB. A Figura 4.9 ilustra o diagrama de classes que representa o *BluetoothConverter*.

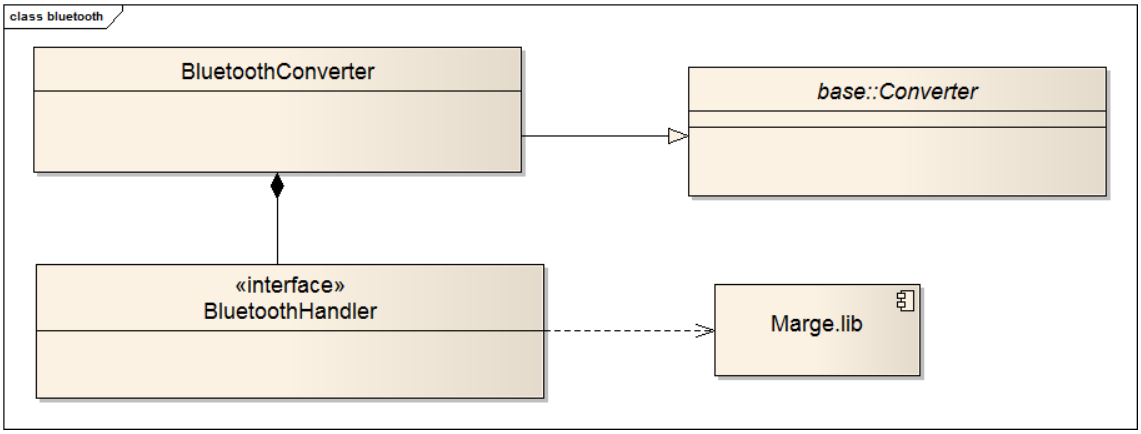

Figura 4.9. Diagrama de Classes do Converter *BluetoothConverter*.

Para lidar com a complexidade da especificação JSR 82, que provê uma complexa interface para utilização do protocolo Bluetooth na plataforma Java, foi utilizado o *framework* Marge [Ghisi 2008]. O *framework* Marge abstrai os conceitos de baixo nível envolvendo a tecnologia *Bluetooth*.

O *BluetoothConverter* ,por meio da biblioteca Marge, realiza descobrimento de dispositivos e recupera o metadado e descrição dos serviços. Dessa forma, os

dispositivos descobertos são convertidos para o domínio *DeviceConverter* e inseridos na *cache* de dispositivos virtuais. Além disso, o componente *BluetoothConverter* transforma as requisições de invocação provenientes da *Bridge* para o formato da tecnologia *Bluetooth*.

4.6 Considerações Finais

A implementação da infra-estrutura de *middleware Device Service Bus* possibilitou que dispositivos de tecnologias heterogêneas pudessem ser conectados a um barramento de comunicação. O DSB tornou esses equipamentos disponíveis para serem anunciados na rede e utilizados por clientes. Além disso, a implementação do DSB pode ser implantada em equipamentos com recursos computacionais limitados, como PDAs e *Smartphones*, desde que suportem a máquina virtual Java CDC 1.2.

Os dispositivos utilizados para construção dos *Converters* não provêem todos os metadados definidos pela especificação DPWS, entretanto os atributos omitidos são opcionais na especificação.

No capítulo seguinte serão apresentados os testes realizados com a infra-estrutura de *middleware Device Service Bus*. Resultados dos testes de desempenho serão apresentados e analisados envolvendo dispositivos conectados ao DSB.

5 Resultados Experimentais

Este capítulo disserta a respeito de todos os experimentos efetuados com o DSB. Para realização dos testes foram utilizados três equipamentos, sendo dois de tecnologias distintas RFID e sensores Sun SPOT. Dois cenários de aplicação real para implantação do DSB serão analisados visando demonstrar como a arquitetura de integração proposta pode ser utilizada para agregar valor a soluções que envolvam uma gama de dispositivos heterogêneos.

5.1 Cenários de Utilização

Para realização dos experimentos foram montados dois ambientes de implantação que consistiram de estações de trabalho atuando como clientes e provedores de serviços e equipamentos conectados às estações provedoras de serviços. As estações de trabalho executavam o Sun Java Toolkit 1.0 para CDC. Dispositivos com suporte a Java ME CDC, como PDAs, *Smartphones* e *Set-top boxes* podem executar DPWS nativamente, não sendo necessário ter um *converter* para interagir com outros dispositivos.

5.1.1 Cenário 1: *Smart Cabinet*

O equipamento *Smart Cabinet*, demonstrado na Figura 5.1, é um protótipo projetado para rastrear equipamentos médicos armazenados em um armário. Equipamentos médicos são responsáveis pelo maior custo dos suprimentos em um hospital, por isso é muito importante rastrear o uso desses equipamentos. Cada

equipamento médico armazenado no *Smart Cabinet* tem uma etiqueta RFID passiva anexada a ele. Então, se algum equipamento for adicionado ou removido, o *Smart Cabinet* consegue saber que equipamento foi adicionado ou removido e a hora exata da ação. Além disso, o acesso a um equipamento médico pode ter restrição a alguma equipe médica, na qual cada profissional estará identificado por uma etiqueta passiva RFID, que pode estar anexada a um crachá. Além disso, se um equipamento médico for adicionado ou removido, o *Smart Cabinet* sabe quem realizou a ação. Qualquer usuário, por meio de uma consulta, pode saber quais equipamentos estão disponíveis naquele momento. Uma lista de equipamentos pode ser consultada remotamente por meio de um PDA, *Smartphone*, ou qualquer outro dispositivo conectado à rede.

Aplicações podem enviar requisições de leitura ao dispositivo leitor RFID Mercury M5 ou M5e, que estariam expostos na rede por meio do DSB. Dessa forma, essas aplicações podem manter em uma *cache* os identificadores dos equipamentos médicos. Essa *cache* pode ser consultada por qualquer outro dispositivo. Se um equipamento médico for removido do *Smart Cabinet* , fornecedores podem ser avisados para que possam repor o equipamento e realizar a cobrança pelo seu uso, caso este seja consignado.

O ambiente montado para execução dos testes é ilustrado na Figura 5.1. O protótipo também foi testado com Sun Java Toolkit 1.0 para CDC. O cenário foi constituído de um *cabinet* comum, identificado pelo número 1, com três cavidades para armazenamento de equipamentos médicos, etiquetas RFID passivas modelo ALN-9540, identificadas pelo número 2, um leitor RFID Mercury M5, identificado pelo número 3 e uma estação de trabalho, identificado pelo número 4, executando Sun Java Toolkit 1.0

para CDC. Uma segunda estação de trabalho estava conectada à mesma rede local, atuando como cliente.

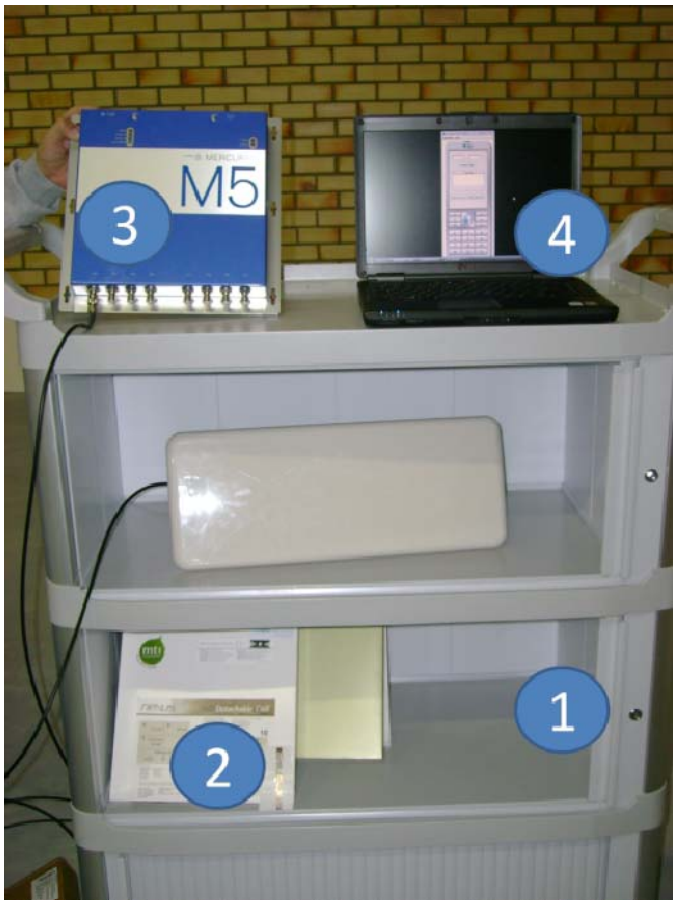

Figure 5.1 – Plataforma de Testes - Equipamento *Smart Cabinet*

5.1.2 Cenário 2: Transporte de Produtos

Muitas empresas transportadoras lidam com produtos de alto valor agregado e a integridade destes itens é uma constante preocupação. Em muitos casos, itens são roubados durante o trajeto e a falta de algum produto só é percebida na chegada ao destino, causando prejuízos às transportadoras.

Visando diminuir e até extinguir as atividades ilegais no transporte de produtos, é possível combinar tecnologias como RIFD e sensores para dar maior rastreabilidade

aos conteúdos dos contêineres. Com a tecnologia RFID é possível saber quais produtos estão ou não em um determinado contêiner. Entretanto, não é possível detectar a remoção de produtos somente com RFID. Dessa forma, pode-se utilizar sensores para detectar alterações de luminosidade no interior do contêiner ou a movimentação da porta. Os administradores dos sistemas das transportadoras, ou até mesmo o motorista, podem ser avisados que algo está errado recebendo um alerta disparado pelos dispositivos no interior do veículo. Um sensor Sun SPOT pode ficar rastreando alterações na luminosidade, temperatura ou movimentação da porta do contêiner. Caso ocorra algum evento externo, como a abertura da porta, o sensor pode enviar uma mensagem por meio do DSB aos Leitores RFID para que seja feita a leitura das *tags* RFID dos produtos, e fazer uma comparação com uma lista de produtos armazenados no contêiner, obtida imediatamente antes do fechamento da porta e mantida em um banco de dados. Se na comparação houver alguma divergência, a administração central e/ou o motorista podem ser avisados. Além disso, as transportadoras podem ter sistemas de informação que busquem, por meio do DSB, informações em tempo real sobre as condições no interior dos contêineres, como temperatura, luminosidade e quais produtos estão armazenados no momento.

No ambiente de teste criado para simular esse cenário, demonstrado na Figura 5.2, o componente identificado com o número 1 é o sensor Sun SPOT, enquanto o componente número 2 é a estação de base, que está ligada via cabo USB ao computador número 3, que é responsável por conectar o Sun SPOT ao computador. O componente 4 é o leitor RFID Mercury M5e, que está conectado via cabo USB ao computador 5, e os componentes identificados com o número 6, são *tags* passivas modelo RR. Os

componentes do DSB e a *RemoteBridge* estavam distribuídos nos componentes 3 e 5, respectivamente; e o cliente em uma outra máquina conectada à mesma rede local.

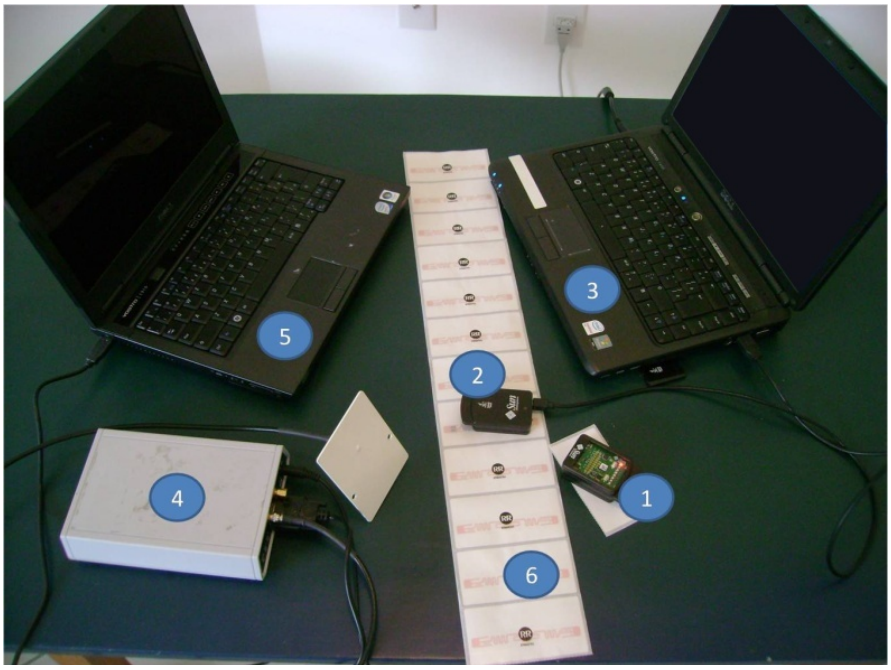

Figura 5.2. Plataforma de Testes – Cenário 2

5.1.3 Características do Ambiente de Testes

Todos os testes foram realizados com duas e três estações de trabalho. Com dois computadores o intuito é simular o ambiente em um computador hospedando o DSB com dispositivos conectados e outro computador hospedando o cliente. Com três computadores a intenção é simular o ambiente com um computador hospedando a *RemoteBridge* com dispositivos conectados, outro computador hospedando o DSB e o terceiro hospedando o cliente. A configuração das estações de trabalho e dos equipamentos foram as seguintes:

- Cliente implementado utilizando *Sun Java Toolkit 1.0 for CDC*, sendo executado em uma máquina com processador Pentium 4 com 2.4 GHz, 512 MB de Memória e sistema operacional Windows XP.
- DSB (*Local Bridge*) implementado utilizando Java SE 1.5, executado em um processador Intel Core 2 Duo 2.0 GHz com 2GB de memória e sistema operacional Windows Vista.
- DSB (*Remote Bridge*) implementado em Java SE 1.5, executado em processador Intel Core 2 Duo 2.0 GHz com 2GB de memória e sistema operacional Windows XP.
- Pilha DPWS com WS4D estendido.
- Roteador Wireless 802.11g com velocidade de 54Mbps.
- Três tipos de *tags* RFID passivas, modelos UMP RAFLATAC *shortdipole*, ALN-9540 e RR;
- Leitor RFID modelo Mercury M5e;
- Leitor RFID modelo Mercury M5;
- Dois sensores Sun SPOT e uma estação-base com SPOT SDK versão 4.0;

Na próxima seção serão apresentados cada medição e a sua respectiva análise. Para todos os experimentos foram adicionados ao DSB dispositivos virtuais simulados com intuito de carregar a *cache* de dispositivos virtuais com 100 dispositivos virtuais, considerando os dispositivos reais e simulados.

5.2 Resultados Obtidos

Foram realizados experimentos nos cenários descritos na seção anterior, com o objetivo de analisar a capacidade do *Device Service Bus* de lidar com vários dispositivos virtuais e requisições simultâneas. Além disso, foi medido o tempo de resposta para descobrimento de dispositivos e serviços e para invocações de serviços do tipo requisição-resposta. Também foram realizadas medições de footprint do DSB, sendo que o DSB com *Bridge* Local consumiu em média 505.508 Bytes e o DSB com Bridge Remota consumiu em média 402.404 Bytes.

5.2.1 Testes com Leitores RFID usando a Bridge Local

Esse experimento foi realizado utilizando a plataforma de teste *Smart Cabinet* , ilustrada pelaFigura 5.1, considerando a configuração de duas estações de trabalho e os leitores RFID. Uma estação hospedando o DSB (*Bridge* Local) com o leitor RID Mercury M5e conectado a essa estação via cabo serial e o leitor RFID Mercury M5 conectado à rede e outra estação hospedando o cliente, todos em uma mesma rede local.

O leitor RFID Mercury M5e disponibiliza uma interface de comunicação por meio de conexão serial, enquanto o leitor RFID Mercury M5 é conectado diretamente à rede Ethernet. Para cada rodada de testes, foram realizadas 100 execuções e obtida a média dos tempos de resposta.

Cada *Virtual Device*, representando tanto o leitor RFID Mercury M5 e RFID Mercury M5e, tinha um *Virtual Service* e este tinha uma *Virtual Action* com um parâmetro de entrada e um de saída. Os leitores RFID tinham um serviço que efetua a leitura de etiquetas passivas e retorna o conteúdo das etiquetas lidas. Inicialmente foi mensurado o tempo de resposta para localizar um dispositivo. Para o processo de

descobrimento foi considerado o tempo para enviar a requisição *probe*, a resposta *probe match*, pegar a descrição do dispositivo e do serviço com as requisições *Get* e *GetResponse*. O tempo médio de resposta para localizar e obter as descrições dos dois leitores e dos seus serviços através da pilha DSB foram de 35,8 ms.

É importante notar que o processo de descobrimento envolve três trocas de mensagens através da rede, como mostra a Figura 3.4. O DSB tinha que encontrar os *Virtual Devices* em sua *cache* para enviar corretamente as mensagem de *probe match*, metadados do dispositivo e descrição dos serviços. Uma requisição é feita por SOAP-sobre-UDP *multicast* para descobrir os dispositivos e são necessárias outras duas requisições SOAP-sobre-HTTP para obter a descrição do dispositivo e do serviço.

Em um teste posterior foi avaliado o tempo de invocação do serviço, variando a quantidade de dados retornados pelos leitores RFID. Na medição do tempo de invocação foi considerado somente o tempo de invocação do *Virtual Action* e o recebimento da resposta, que variou de 10 *KBytes* a 100 *KBytes*. A Figura 5.3 mostra os resultados dos testes. O resultado mostra que o tempo de resposta cresce a uma taxa menor que o tamanho da resposta. A medição do tempo levou em consideração o processamento realizado pelo DSB para processar requisições SOAP-sobre-HTTP e despachá-las para o dispositivo correto, além o próprio processamento realizado pelo dispositivo provedor de serviço. Isso mostra que o mecanismo de invocação tem funcionamento satisfatório para serviços que tem uma alta taxa de transmissão de dados.

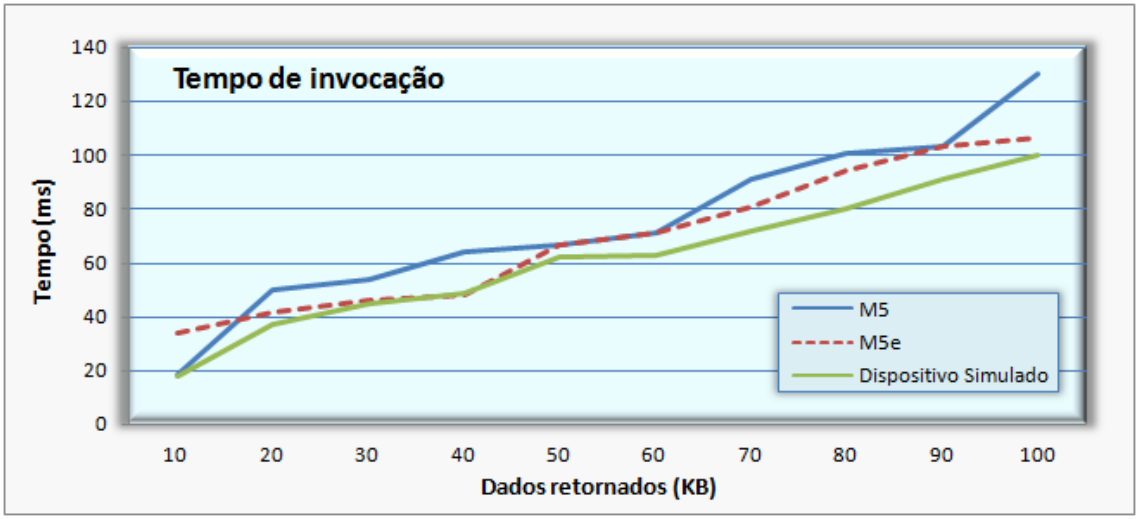

Figura 5.3. Tempo de Invocação Utilizando DSB – *Bridge* Local

5.2.2 Testes com Leitores RFID usando a Bridge Remota

Para esse experimento também foi utilizada a plataforma de teste da Figura 5.1, considerando a configuração de três estações de trabalho e os leitores RFID. Uma estação hospedando o DSB (*Bridge* Remota) com o leitor RID Mercury M5e conectado a essa estação via cabo serial e o leitor RFID Mercury M5 conectado à rede, outra estação hospedando o DSB (*Bridge* Local) e outra estação hospedando o cliente, todos em uma mesma rede local. Diferenciando apenas na configuração de implantação, esse experimento teve as mesmas características que o experimento descrito na seção anterior.

Como demonstrado nas Figuras 3.10 e 3.11, o fluxo da requisição invocada pelo cliente é efetuada para a *Bridge* Local e esta a repassa para a *Bridge* Remota, na qual os dispositivos estão conectados. Em virtude dessa configuração de implantação, o tempo de resposta aumentou, como mostra a Figura 5.4. Entretanto, a distribuição de dispositivos utilizando o DSB e a *Bridge* Remota ainda se mostra satisfatória, pois um

aparelho, por exemplo um PDA, pode hospedar a pilha DSB e não prover meios de comunicação como cabo serial. Dessa forma, a utilização de uma *BridgeRemota* possibilita que mais dispositivos possam ser expostos e utilizados por meio do *Device Service Bus*.

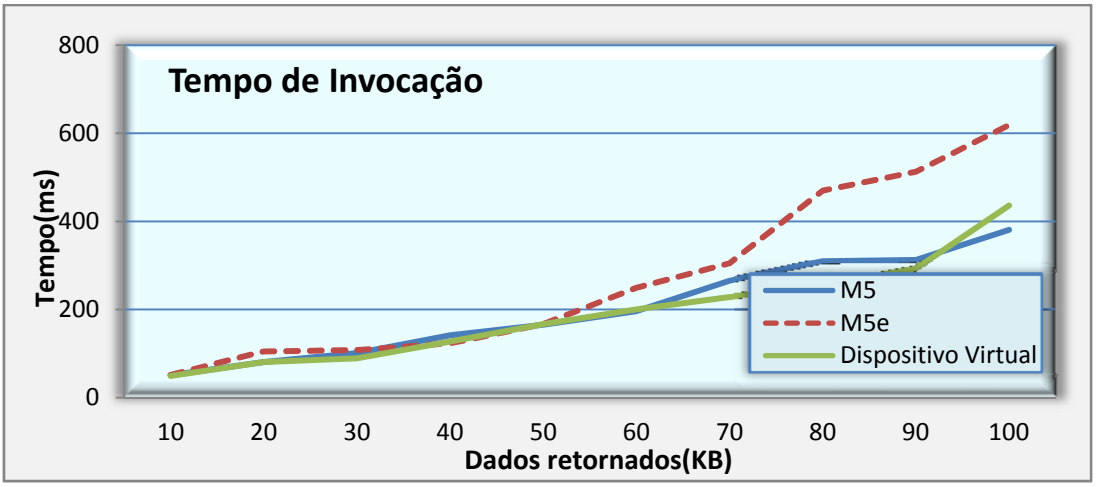

Figura 5.4. Tempo de Invocação Utilizando DSB – *Bridge* Remota

5.2.3 Testes com os sensores Sun SPOT usando *Bridge* Local e Remota

Esta seção aborda dois experimentos executados com os sensores Sun SPOT. Para estes dois experimentos também foi medido o tempo de resposta da invocação de um *VirtualAction*, com um parâmetro de saída que representada a intensidade de luz. Para cada rodada de testes, foram realizadas 100 execuções e obtida a média dos tempos de resposta. A configuração de implantação de um dos experimentos consistiu em um computador hospedando o DSB (*Bridge* Local) com a estação-base Sun SPOT conectada via cabo USB e um sensor Sun SPOT conectado via WPAN à estação-base e

outro computador hospedando um cliente. A configuração de implantação do segundo experimento constituiu em um computador hospedando o DSB (*Bridge* Remota) com a estação-base Sun SPOT conectada via cabo USB e um sensor Sun SPOT conectado via WPAN à estação-base, outro computador hospedando o DSB (Bridge Local) e um terceiro computador hospedando um cliente.Todos os computadores estavam na mesma rede local.

Para o caso do experimento com o Sun SPOT foi considerado que o dispositivo retornasse dados de 10 a 100 Kbytes. Entretanto, o Sun SPOT utiliza conexão de rádio (WPAN) que permite o envio de pacotes com até 1.2 KBytes. Então, para poder retornar uma quantidade de dados maior, foram feitas várias requisições entre a estação-base e o Sun SPOT, enquanto cliente somente realizava uma requisição.

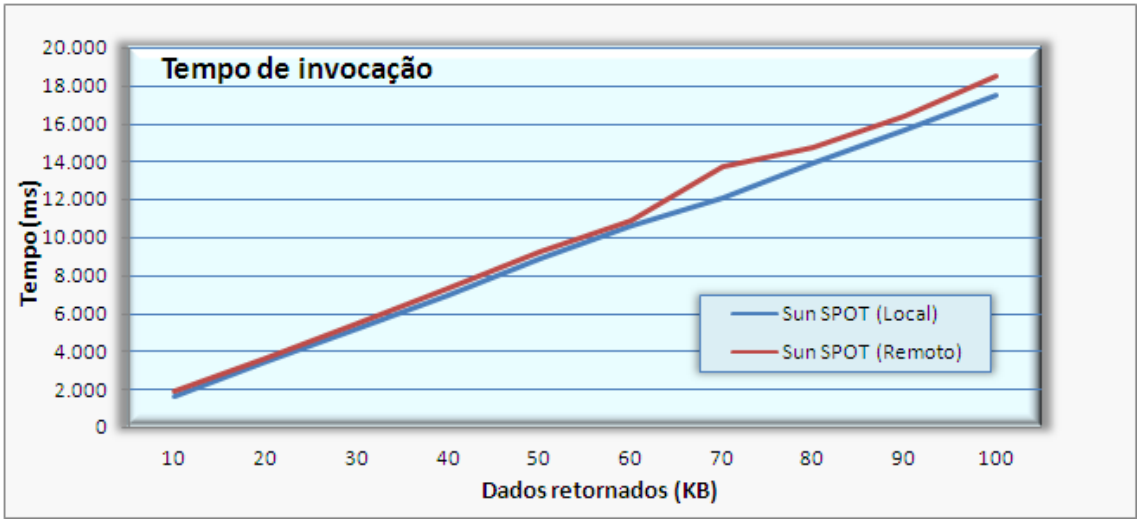

Figura 5.5. Tempo de Invocação do Sun SPOT

Os resultados obtidos com o Sun SPOT, Figura 5.5, mostram que o tempo de resposta da invocação do serviço aumenta de acordo com a quantidade de dados retornados em uma proporção e dois segundos para cada dez *kilobytes* de dados transmitidos. Além disso, a tempo consumido pelo uso da *Bridge* Remota não

demonstrou uma grande diferença em relação a *Bridge* Local, podendo concluir que o maior tempo consumido foi realizado pelo dispositivo para processamento da requisição.

5.2.4 Testes com Sun SPOT e Leitor RFID usando *Bridge* Local e Remota

Esta seção apresenta dois outros experimentos realizados com Sun SPOT e o leitor RFID Mercury M5e. A configuração de implantação de um dos experimentos constituiu em um computador hospedando o DSB (*Bridge* Local) e conectado ao leitor RFID Mercury M5e e à estação-base Sun SPOT, que por sua vez conecta-se via WPAN a um sensor Sun SPOT. A configuração de implamantação do segundo experimento constituiu em um computador hospedando o DSB (*Bridge* Local) e conectado a ele a estação-base SunSPOT e conectado a estação-base via WPAN um sensor Sun SPOT e um segundo computador hospedando a *Bridge* Remota e conectado a ele o leitor RFID Mercury M5e.

O objetivo desses experimentos foi ter o dispositivo Sun SPOT como cliente em busca de outros dispositivos e serviços, nesse caso o leitor RFID M5e. A partir de um evento físico, que foi a variação na intensidade de luz, o sensor Sun SPOT invocava o DSB para que fosse executado o serviço de leitura de etiquetas do leitor RFID.

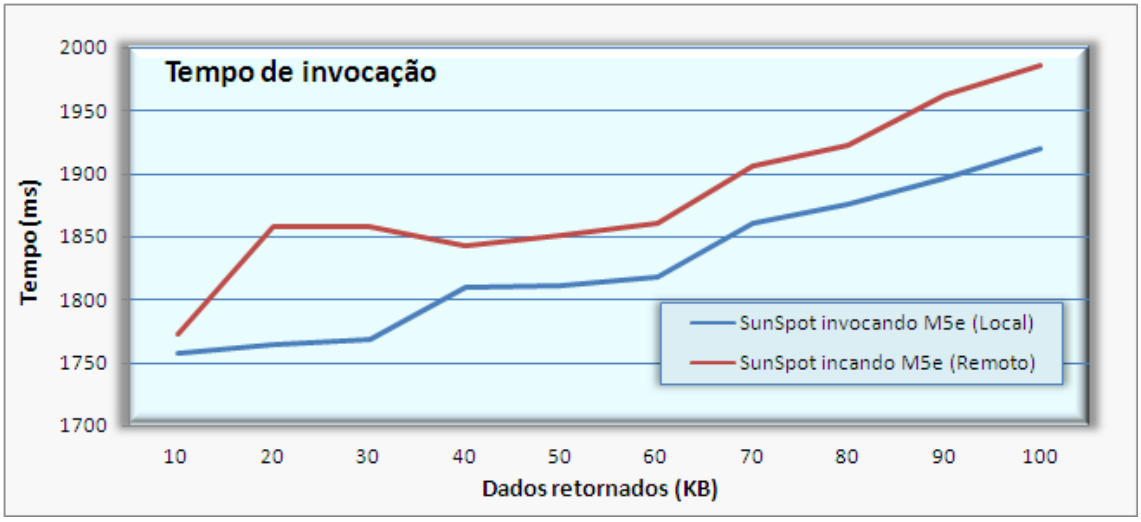

Figura 5.6. Tempo de Invocação do Sun SPOT como cliente do Leitor RFID M5e.

Os resultados obtidos com os dois experimentos nas configurações citadas, Figura 5.6, mostram que as invocações realizadas pelo Sun SPOT ao leitor RFID Mercury M5e tem uma tendência de crescimento com o aumento do dados retornados pelo leitor RFID.

A próxima seção descreve o último experimento com o DSB, *Bridge* Remota e o sensor Sun SPOT. Nesse caso, o objetivo é medir o tempo de cada etapa da invocação, assim será possível visualizar quanto tempo cada ator do experimento gasta para executar uma invocação requisição-resposta.

5.3 Medição das Etapas da Requisição ao *Device Service Bus*

Enfim, estes quatro últimos experimentos têm o objetivo de proporcionar uma visão de cada etapa da execução do *Device Service Bus*. Dessa forma é possível

visualizar quanto tempo é gasto pelo DSB e *Bridge* remota independente do dispositivo que estiver conectado a ele.

Dois experimentos foram realizados com o Sun SPOT, retornando 20kbytes de dados, com a mesma metodologia e configuração de implantação utilizada nos experimentos do capítulo 5.3. A Tabela 2 mostra a realização somente com o DSB e a Tabela 3 com o DSB e *Bridge* Remota.

	Tempo (ms)	**% de Participação**
Transmissão de Rede	7,6	2,8
DSB	27	7,2
Sun SPOT	338	90

Tabela 2. Tempo Particionado entre DSB, Sun SPOT e Transmissão de Rede

	Tempo (ms)	**% de Participação**
Transmissão de Rede	60	13,1
DSB	10	2,8
Bride Remota	28	6,1
Sun SPOT	358	78

Tabela 3. Tempo Particionado entre DSB, *Bridge* Remota, Sun SPOT e Transmissão de Rede

Os dois experimentos com Sun SPOT mostram que a maior parcela de processamento na execução de serviços no dispositivo está relacionada à tecnologia conectada ao *Device Service Bus*. O tempo de transmissão de rede e dos participantes DSB e *Bridge* Remota tendem a ser menores que o tempo de processamento do dispositivo, mesmo variando a quantidade de dados retornados.

Os outros dois experimentos foram realizados com o M5e, retornando 20KBytes de dados, nas mesmas configurações de implantação das sessões 5.2.1 e 5.2.2. A Tabela 4 mostra a realização somente com o DSB e a Tabela 5 com o DSB e *Bridge* Remota.

	Tempo (ms)	**% de Participação**
Transmissão de Rede	3,4	7,8%
DSB	27	62%
M5e	13	30,2%

Tabela 4. Tempo Particionado entre DSB, M5e e Transmissão de Rede

	Tempo (ms)	**% de Participação**
Transmissão de Rede	26	25,1
DSB	10	9,5
Bride Remota	28	26,4
M5e	42	39

Tabela 5. Tempo Particionado entre DSB, *Bridge* Remota, M5e e Transmissão de Rede

Os dois experimentos com M5e mostram que com a *Bride* Local a maior parcela de processamento está no DSB, pois a leitura do conteúdo das etiquetas é feita a partir de uma *cache*, armazenada no dispositivo. Não é considerado o tempo de leitura da etiqueta, pois esse tempo é um parâmetro variável a ser passado para o leitor RFID, que nesse experimento foi de 250 milisegundos. Para o experimento com a *Brigde* Remota, a maior parcela de tempo está no dispositivo, e o tempo do DSB diminui nesse caso, pois com o uso da *Bridge* Remota o *Converter* utilizado está na *Bridge* Remota, dessa forma o processamento do *Converter* é realizado remotamente.

Os experimentos com *BluetoothConverter* ainda não foram realizados, pois o seu desenvolvimento não está concluído. O desenvolvimento do conversor está sendo realizado como parte do trabalho de conclusão de curso do aluno Fábio Kreush, acadêmico do curso de Sistemas de Informação da Universidade Federal de Santa Catarina (UFSC).

6 Conclusões

O presente trabalho mostrou que foi possível projetar uma arquitetura de integração para lidar com dispositivos de classes diversas. Disponibilizar serviços contidos em dispositivos, realizando anúncios na rede é a característica *plug and play* provida pela arquitetura para tornar a configuração para uso dos dispositivos menos dependente de intervenção de usuários. Com essa proposta de trabalho também se atingiu o objetivo de prover aos equipamentos, formas de busca por serviço e execução de ações contidas nos serviços.

Como resultado do trabalho, foi desenvolvido o *Device Service Bus* (DSB), uma infra-estrutura de *middleware* para integração de dispositivos heterogêneos que é baseado na especificação DPWS (*Device Profile for Web Service*). A especificação DPWS segue o padrão consagrado da tecnologia de serviços web para integração de sistemas em ambiente coorporativo, trazendo para a realidade da computação embarcada os benefícios do paradigma da arquitetura orientada a serviços e a utilização de protocolos de rede largamente utilizados como HTTP, TCP, UDP *multicast* e *unicast* e SOAP.

O DSB integra dispositivos que empregam diferentes tecnologias de comunicação, como RFID, Bluetooth e Sun SPOT, permitindo que dispositivos e serviços sejam expostos como serviços web. Além disso, as interfaces dos equipamentos e dos seus serviços podem ser disponibilizadas em vários pontos de acesso, além de conversores de tecnologias conectados remotamente ao ponto de acesso.

A solução proposta para integração de dispositivos é mais leve que outras soluções propostas encontradas na literatura, sendo capaz de executar tanto em equipamentos móveis como PDA e *Smartphones*, como remotamente em uma estação de trabalho à qual o dispositivo a ser integrado tenha acesso, ampliando dessa forma os tipos de nodos que proporcionam acesso a outros dispositivos. Além disso, dispositivos com recursos limitados como sensores e etiquetas RFID podem ser integrados através de dispositivos de interconexão, que hospedem *Bridges* e *Converters* de tecnologias específicas.

Enfim, pode-se analisar que com o DSB é possível compor diversos dispositivos para proporcionar soluções para diferentes cenários de aplicação. Dessa forma, aproveitando tecnologias heterogêneas com papéis específicos, agrega-se valor a projetos que necessitem de soluções integradoras.

6.1 Limitações do Trabalho

O trabalho se concentrou em prover o barramento de comunicação entre tecnologias, sendo projetada uma arquitetura que pudesse ser expansível para dar continuidade à construção de *Converters*. Entretanto, foram escolhidas somente três tecnologias para dar início e validar a infra-estrutura proposta.

Não foram abordados no projeto aspectos de segurança na utilização de serviços. Contudo, a especificação DPWS comporta o *ws-security* que provê aspectos de autenticação e criptografia de dados. A garantia na troca de mensagem também não foi objeto de estudo para esse projeto, porém a especificação DPWS comporta o *ws-policy*, dessa forma é possível definir políticas para execução de serviços.

6.2 Trabalhos Futuros

Atualmente, estão sendo finalizados os testes do *Converter* para Bluetooth e a inicialização de *Converter* para dispositivos GPS. No futuro, planeja-se trabalhar no desenvolvimento de outros *Converters* para incorporar mais dispositivos e outras tecnologias de comunicação ao DSB, com a intenção de ter diferentes classes de dispositivos em um ambiente ubíquo e heterogêneo.

Além disso, pretende-se trabalhar com o *framework* para .Net, que implementa a especificação DPWS, para implementação da arquitetura do *Device Service Bus*, assim, provendo em outra plataforma a solução de interoperabilidade para dispositivos diversos. Também planejasse, adicionar características de qualidade de serviço (QoS) para busca e invocação de serviço utilizando a especificação *ws-policy* suportada pelo DPWS, além de poder realizar buscas semânticas. Outro aspecto a ser estudado é a implementação das características de segurança utilizando a especificação *ws-security*, também suportada pela especificação DPWS.

7 Referências Bibliográficas

ALLARD, J.,; CHINTA, V.; GUNDALA, S.; RICHARD, G. G. III: *Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability*. *Proceedings of the Symposium on Applications and the Internet*, 2003, Page(s) 268-275.

BOHN, Hendrik; BOBEK, Andreas; GALATOWISK, Frank. SIRENA - *Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains.* IEEE *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL)*, 2006, Page(s) 43-43.

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

CHATSCHIK, B. An Overview of the Bluetooth Wireless Technology. IEEE Communications Magazine, 2001. Vol. 39(12), Page(s) 86-94.

DELPHINANTO A.; LUKKIEN, J.J.; KOONEN; et al: *Architecture of a bidirectional Bluetooth-UPnP proxy*. IEEE *Consumer Communications and Networking Conference*, 2007. CCNC. 4th, , Page(s) 34-38.

EDWARDS, W. Keith. *Discovery Systems in Ubiquitous Computing*, 2006. *Pervasive Computing* IEEE CS *and* IEEE ComSoc - 1536-1268, p 70-77.

ERL T., *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*, Prentice Hall, 2004.

FUHRER, Patrik; GUINARD, Dominique ;LIECHTI, Olivier: RFID: From Concepts to Concrete Implementation, 2006. IPSI-2006 MARBELLA, *February 10-13, 2006, Marbella, Spain.*

GHISI, Bruno: MARGE DEV, 2008. Disponível em <https://marge.dev.java.net/>.

HOWITT, I.; GUTIERREZ, J.A: IEEE 802.15.4 low rate - wireless personal area network coexistence issues. Wireless Communications and Networking (WCNC). March 2003 Page(s):1481 - 1486 vol.3

HWANG,Taein; PARK,Hojin; CHUNG, Jin Wook: *A Study on UPnP A/V Session Mobility Based on RFID*. IEEE *Advanced Communication Technology*, 2008 (ICACT) Page(s):1801 - 1802 vol.3.

JAMMES, François; MENSCH, Antoine; SMIT, Harm: *Service-oriented device communications using the devices profile for web services. Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*,2005. ACM Vol. 115.

LOUREIRO, Antônio; NOGUEIRA, José; RUIZ, Linnyer, et al: Redes de Sensores Sem Fio. XXI Simpósio Brasileiro de Redes de Computadores ,2003. Belo Horizonte, Minas Gerais.

MACHADO, Guilherme Bertoni: Integration of Embedded Devices Through Web Services: Requirements, Challenges and Early Results, 2006. Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06). Calgary, Italy.

MICROSOFT Corporation. Devices Profile for Web Services (DPWS). 2006. Available at http://schemas.xmlsoap.org/ws/2006/02/devprof/.

MICROSOFT Corporation. The Web Services Dynamic Discovery (WS-Discovery), 2005. Available at http://specs.xmlsoap.org/ws/2005/04/discovery/.

OSGi Alliance. *OSGi Service Platform Release 4 CORE*, 2005. Available at http://www.osgi.org/Specifications/HomePage.

PRABAKAR, V; KUMAR, Dr. BV; SUBRAHMANYA, SV: *Management of RFID-centric business networks using Web Services*. IEEE *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services,* 2006, Page(s) 133-133.

RAVERDY, Pierre-Guillaume; RIVA, Oriana; CHAPELLE, Agnès; et al: *Efficient Context-aware Service Discovery in Multi-Protocol Pervasive EnvironmentS,* 2006. *Proceedings of the IEEE 7th International Conference on Mobile Data Management* (MDM'06), Page(s) 3-3.

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

SCHALL D., AIELLO M., DUSTDAR S. We Service on Embedded Devices. International Journal of Web Information Systems (IJWIS), 2006.Troubador Publisher, vol 2 Page(s) 45-50.

SIEGEMUND, F., FLÖRKEMEIER, C: *Interaction in Pervasive Computing Settings using Bluetooth-Enabled Active Tags and Passive RFID Technology together with Mobile Phones*. Proc. IEEE PerCom, 2003, Page(s) 378-387.

SUN Microsystems. Especificações JINI Archive, 2005. v2.1. Available at http://java.sun.com/products/jini/2_1index.html.

SUN Microsystems: Getting Started with Sun SPOT, 2008. Available at http://www.Sun SPOTworld.com/docs/.

UPNP Forum. Arquitetura de Dispositivo UPnP v1.0, 2006. Available at http://www.upnp.org/resources/documents.asp.

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

W3C. Arquitetura de Serviço Web. 2004. Available at http://www.w3.org/TR/ws-arch/.

WANT. R. An Introduction to RFID Technology. IEEE Pervasive Computing. 2006, Vol 5(1), Page(s) 25-33.

WEISER, Mark. Tópicos: Ubiquitous Computing. IEEE Computer, 1993. 26(10):71–72.

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

YIM, Hyung-Jun; OH, Il-Jin; HWANG, Yun-Young;et al: *Design of* DPWS *Adaptor for Interoperability between Web Services and* DPWS *in Web Services on Universal Networks,* 2007.*Proceedings of the* IEEE *International Conference*, Page(s) 1032-1039.

ZEEB, Elmar; BOBEK, Andreas; BOHN, Hendrik; PRÜTER, Steffen; et al:. *The Third International Conference on Open Source Systems,* 2007. Limerick, Ireland.

Zigbee™ Alliance, Sítio: HTTP://www.zigbee.org.