Universidade Federal de Santa Catarina

# Integração de Dispositivos Móveis com Web Services através de Bluetooth

Fabio Kreusch

**Monografia de Conclusão do Curso de Sistemas de Informação**

Orientador:
Prof. Frank Augusto Siqueira, Dr.

Florianópolis, julho de 2009.

Fabio Kreusch

# Integração de Dispositivos Móveis com Web Services através de Bluetooth

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Frank Augusto Siqueira, Dr. - Orientador

Florianópolis, 2009

*Dedico este trabalho à minha família.*

# AGRADECIMENTOS

*Agradeço a todos que direta ou*
*indiretamente participaram de minha*
*formação como pessoa e profissional.*

*A mente que se abre a uma nova ideia jamais*
*voltará ao seu tamanho original.*

***Albert Einstein***

# RESUMO

Este trabalho de conclusão de curso visa estudar formas de interação entre sistemas no contexto ubíquo da tecnologia Bluetooth. Para isto, tecnologias de integração como serviços web são analisadas e uma camada de integração entre a tecnologia Bluetooth e uma arquitetura de serviços web é desenvolvida. O desenvolvimento desta camada de integração visa permitir que dispositivos de recursos limitados possam fazer parte de um ambiente ubíquo integrado, independente das tecnologias de comunicação envolvidas.

Palavras-chave: Serviços web; Bluetooth; Ambiente ubíquo; Dispositivos móveis.

# ABSTRACT

This work aims to study ways to integrate systems in the ubiquitous context of the Bluetooth technology. For that, integration technologies like web services are analyzed and an integration layer between the Bluetooth technology and a web services architecture are developed. The developing of this integration layer aims to make possible that limited devices be part of an integrated ubiquitous environment, independently of the used technologies.

Keywords: Web services; Bluetooth; Ubiquitous environment; Mobile devices.

# LISTA DE FIGURAS

Figura 2.1: Visão geral de SOA (W3C, 2009b)....................................................................16

Figura 2.2: Exemplo de um arquivo XML...............................................................................20

Figura 2.3: Exemplo de um arquivo DTD...............................................................................20

Figura 2.4: Exemplo de um XML Schema..............................................................................22

Figura 2.5: Envelope SOAP.....................................................................................................23

Figura 2.6: Exemplo de um WSDL..........................................................................................24

Figura 3.1: A pilha de protocolo Bluetooth (APPLE2, 2008b)...............................................28

Figura 3.2: Redes Bluetooth.....................................................................................................31

Figura 4.1: O protótipo do Green Project (SUN, 2008b).........................................................33

Figura 4.2: Duke, o mascote (JAVANET, 2008).....................................................................34

Figura 4.3: As edições da plataforma Java...............................................................................35

Figura 5.1: Diferentes dispositivos em conjunto......................................................................38

Figura 5.2: Comunicação proposta pelo DPWS (MSDN, 2008b)...........................................40

Figura 5.3: A arquitetura DSB..................................................................................................41

Figura 5.4: Dinâmica de busca de dispositivos e serviços.......................................................42

Figura 5.5: Dinâmica de invocação de um serviço...................................................................43

Figura 5.6: Arquitetura dos componentes envolvidos..............................................................43

Figura 6.1: Diagrama de classes do BluetoothConverter.........................................................47

Figura 6.2: Classes do pacote br.ufsc.inf.ppgcc.converter.bluetooth.......................................49

Figura 6.3: Arquitetura do framework (MARGE, 2008b)........................................................50

Figura 6.4: Arquitetura Bluecove (BLUECOVE, 2009b)........................................................51
Figura 6.5: Implementação da estrutura Bluetooth..................................................................52
Figura 6.6: Diagrama de sequência de uma busca de serviços.................................................53
Figura 6.7: Converters...............................................................................................................54
Figura 6.8: Invocação de serviço..............................................................................................55
Figura 6.9: Busca de serviços....................................................................................................56
Figura 6.10: XML de busca de serviços....................................................................................57
Figura 6.11: Invocação de serviço.............................................................................................57
Figura 6.12: XML de invocação de serviço..............................................................................58
Figura 6.13: Arquitetura de testes.............................................................................................58
Figura 6.14: Gráfico do tempo de invocação do teste 1...........................................................60
Figura 6.15: Gráfico do tempo de invocação do teste 2...........................................................62

# LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

CORBA – Common Object Request Broker Architecture

DCOM – Distributed Component Object Model

DSB – Device Service Bus

DTD – Document Type Definition

HTML – Hyper Text Markup Language

Java EE – Java Enterprise Edition

Java ME – Java Micro Edition

Java SE – Java Standard Edition

JSR – Java Specification Requests

KB – Kilobytes

L2CAP – Logical Link Control and Adaptation Protocol

OBEX – Object Exchange

OMG – Object Management Group

RFCOMM – Radio Frequency Communication

RFID - Radio-Frequency IDentification

RMI – Remote Method Invocation

SDAP – Service Discovery Application Profile

SOA – Service Oriented Architecture

SOAP – Simple Object Access Protocol

UDDI – Universal Description, Discovery and Integration

URI – Universal Resource Identifier

W3C – World Wide Web Consortium

WPAN - Wireless Personal Area Network

WSDL – Web Service Definition Language

XML – Extensible Markup Language

# LISTA DE TABELAS

Tabela 1: Resultados do teste 1........................................................................................59

Tabela 2: Resultados do teste 2........................................................................................61

# SUMÁRIO

**1 Introdução....................................................................................................13**

*1.1 Objetivos........................................................................................................14*

*1.2 Justificativa....................................................................................................14*

*1.3 Organização do Texto...................................................................................15*

**2 Web services..................................................................................................16**

*2.1 SOA...............................................................................................................16*

*2.2 Definição de Web services............................................................................17*

*2.3 Benefícios......................................................................................................18*

*2.4 Padrões e Tecnologias..................................................................................19*

2.4.1 XML...............................................................................................................20

2.4.2 SOAP.............................................................................................................22

2.4.3 WSDL............................................................................................................23

2.4.4 UDDI..............................................................................................................25

**3 Bluetooth........................................................................................................26**

*3.1 Desafios.........................................................................................................27*

*3.2 Arquitetura.....................................................................................................27*

3.2.1 Camada inferior.............................................................................................28

3.2.2 Camada Superior...........................................................................................29

3.2.3 Perfis Bluetooth.............................................................................................30

*3.3 Topologia de comunicação............................................................................30*

3.3.1 Operações de comunicação..........................................................................31

**4 Java................................................................................................................33**

*4.1 A tecnologia...................................................................................................34*

*4.2 Java APIs for Bluetooth.................................................................................36*

4.2.1 Utilização......................................................................................................37

**5 Integração de dispositivos com web services...............................................38**

*5.1 DPWS – Devices Profile for Web Services....................................................39*

*5.2 DSB – Device Service Bus.............................................................................41*

**6 Integração da tecnologia Bluetooth ao DSB....................................................45**

*6.1 Diagrama de classes...........................................................................................45*

*6.2 Converter...............................................................................................................48*

6.2.1 Marge........................................................................................................................50

6.2.2 Bluecove....................................................................................................................51

6.2.3 BluetoothHandlerMargeImpl.....................................................................................52

6.2.4 Invocação de um Serviço Bluetooth..........................................................................55

6.2.5 Descoberta e invocação de serviços  por clientes Bluetooth......................................56

*6.3 Testes.....................................................................................................................58*

6.3.1 Teste 1: DSB acessando um serviço Bluetooth..........................................................59

6.3.2 Teste 2: Cliente Bluetooth acessando um serviço do DSB.........................................61

6.3.3 Avaliação...................................................................................................................63

**7 Considerações finais e trabalhos futuros........................................................64**

**8 Referências bibliográficas..................................................................................66**

**Anexo A – Artigo.......................................................................................................70**

**Anexo B – Código Fonte...........................................................................................82**

# 1 INTRODUÇÃO

A evolução da computação em todas as suas faces tem criado um novo ambiente propício à comunicação constante entre diversos dispositivos, independentemente de sua localização. Telefones celulares comunicam-se com outros através de ondas eletromagnéticas, computadores comunicam- se entre si e com outros dispositivos através da internet, pequenos dispositivos comunicam-se através de pequenas redes ad-hoc. Estes exemplos compartilham um objetivo fim, o de realizar alguma tarefa para um ser humano, seja executar a comunicação entre pessoas, ou prover alguma funcionalidade que seja útil ao usuário.

Este conceito de comunicação ocorrendo em nosso meio, muitas vezes sem a percepção imediata de que o processo envolve alguma computação, é conhecido como computação ubíqua (*Ubiquitous Computing*), termo cunhado pelo pesquisador Mark Weiser (PALO, 2008), e que vem recebendo atenção cada vez maior por parte da comunidade acadêmica e da indústria.

Aliado a conceitos de intercomunicação, temos também acompanhado o crescimento do estudo e utilização da arquitetura de serviços web. Serviços web (do inglês, *web* services) visam facilitar a troca de informação entre diferentes aplicações (W3C, 2008a), independente da plataforma onde a aplicação é executada (dispositivos móveis ou não, sistemas operacionais, etc) e independente da linguagem em que o aplicativo foi desenvolvido. Para isto, padrões de troca de informação foram definidos, como a utilização do protocolo SOAP e documentos XML.

A monografia se propõe a estudar e especificar mecanismos para comunicação entre dispositivos móveis e baseados em serviços web utilizando o Bluetooth (BLUETOOTH SIG, 2008a), uma tecnologia que tem sido largamente adotada em diversos dispositivos móveis, por suas características de robustez, baixo consumo de energia e baixo custo. Para este propó-

sito, será estendida a arquitetura DSB (ARAUJO SIQUEIRA, 2008), que prevê um meio de intercomunicação entre dispositivos de várias tecnologias. Neste trabalho, uma extensão que permita integrar um dispositivo com suporte a Bluetooth será desenvolvida e adaptada para funcionar junto ao DSB.

## 1.1 Objetivos

Este trabalho tem como objetivo principal desenvolver uma camada de software baseada nas especificações do DSB que seja responsável por realizar a troca de informações com dispositivos baseados na tecnologia Bluetooth.

Para atingir este objetivo, será necessário realizar um estudo da tecnologia Bluetooth e seu suporte na linguagem Java (Java SE – Standard Edition – e Java ME – *Micro* Edition), que será utilizada como linguagem para implementação. Além disso, serão estudados também os padrões existentes de comunicação entre serviços web e a arquitetura DSB.

## 1.2 Justificativa

Diante a necessidade de integração entre diferentes sistemas, a arquitetura de serviços web tem se mostrado uma alternativa viável e que tem resolvido em grande parte os problemas de intercomunicação de sistemas. Serviços dos mais variados tipos estão disponíveis na forma de serviços web.

Dispositivos móveis já fazem parte do dia a dia de várias pessoas, estas os utilizando para diversos propósitos, seja entrar em contato com outras pessoas, acessar uma agenda, ler e-mails, navegar na Web, etc. Muitas destas tarefas podem ser realizadas através de serviços web. O problema é que as tecnologias padronizadas de serviços web muitas vezes não estão implementadas em dispositivos móveis, como os dispositivos com suporte a Bluetooth, parcialmente excluindo-os das facilidades propostas pelos serviços web.

Este trabalho estudará a viabilidade de desenvolver uma camada intermediária de comunicação, que será responsável por integrar dispositivos Bluetooth a um provedor de serviços web.

## 1.3 Organização do Texto

O capítulo 2 apresenta a tecnologia de Serviços Web, suas especificações e objetivos, e implementações existentes da arquitetura.

O capítulo 3 tem como objetivo estudar as características técnicas do Bluetooth.

O capítulo 4 é dedicado a linguagem de desenvolvimento Java e suas plataformas Java SE e Java ME.

Por fim, no capítulo 5 será demonstrada a conceituação e o desenvolvimento da camada responsável por integrar dispositivos Bluetooth a um provedor de serviços web.

# 2 WEB SERVICES

## 2.1 SOA

SOA (*Service-Oriented Architecture)* é um estilo de arquitetura genérico, um paradigma conceitual, que suporta orientação a serviços. Orientação a serviços é uma maneira de desenvolver software, onde existem serviços que podem ser utilizados por clientes e estes retornam uma resposta. De forma geral, um serviço é uma atividade repetível que produz um resultado específico (como por exemplo, validar um número de cartão de crédito) (OPEN GROUP, 2009).

Do ponto de vista de negócios, a arquitetura SOA pode ser definida como “um conjunto de serviços que pode ser utilizado na construção de soluções e que pode ser exposto a clientes e parceiros”. Do ponto de vista de arquitetura de software, pode ser definido como um “conjunto de princípios e padrões para prover as seguintes características a um software: modularidade, encapsulamento, baixo acoplamento, reutilização e composição”. (IBM, 2009)

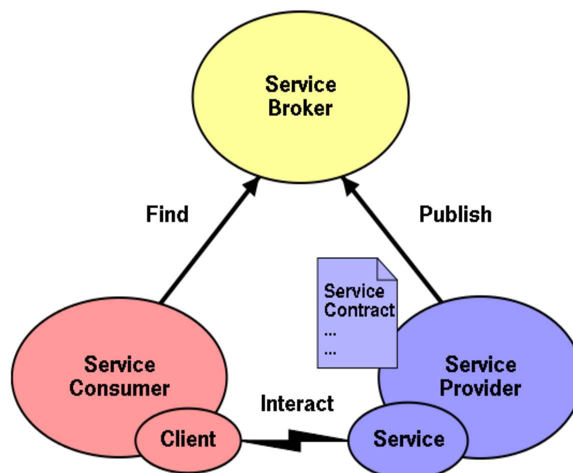



**Figura 2.1: Visão geral de SOA (W3C, 2009b)**

A figura 2.1 apresenta as principais entidades envolvidas na arquitetura SOA. O *Service Provider* é o provedor de um serviço, a entidade que executa um determinado serviço. O *Service Broker* age como um disponibilizador de serviços. *Service Providers* publicam seus serviços em um *Service Broker*, permitindo assim que consumidores encontrem seus serviços listados no *Service Broker*. *Service Consumers* são os consumidores de serviços, que fazem a busca em um *Service Broker* e solicitam a execução dos serviços.

Uma das formas de implementar a intercomunicação entre diferentes serviços através da arquitetura SOA é criar uma interface para a comunicação com estes serviços utilizando Web services.

## 2.2 Definição de Web services

Um *web service* (serviço web) é um software (uma aplicação) acessível através de uma rede (internet, intranet, etc.) e identificado por meio de uma URI (*Universal Resource Identifier*[1]), que é acessada por clientes utilizando protocolos baseados em XML. (SINGH BRYDON et al, 2004).

Web services são a evolução da Web. Inicialmente, a Web era composta primordialmente por páginas HTML estáticas. Posteriormente, aplicações web foram desenvolvidas de forma a gerar páginas dinamicamente, oferecendo a possibilidade de prover conteúdos diferentes aos clientes dependendo das requisições feitas. Por exemplo, inicialmente tínhamos uma lista de serviços contínua listada em uma página estática HTML. As aplicações web permitiram criarmos filtros, onde somente os resultados filtrados são demonstrados no HTML enviado ao cliente.

Porém, este tipo de aplicação web está restrita apenas aos clientes de páginas HTML. Os *web services* são a evolução das aplicações web. Resultados de uma consulta a um web service retornam dados encapsulados no formato XML, um formato de dados especificado para facilitar a interoperabilidade entre aplicações. Desta forma, uma aplicação web pode solicitar a um serviço web um determinado resultado, receber uma resposta XML, e realizar operações diversas a partir dos resultados.

---

**1** Uma URI é um endereço que identifica um recurso em uma rede, sendo imagens, textos, serviços, etc., que tornam um recurso disponível através de um método de acesso, como HTTP ou FTP (W3C, 2008).

*Web services* representam o amadurecimento de conceitos tradicionais de ambientes de computação distribuída, como RMI (*Remote Method Invocation*, uma tecnologia Java para interação de objetos distribuídos em rede), DCOM (*Distributed Component Object Model*, um conjunto de conceitos e interfaces para requisição de serviços em rede proposta pela Microsoft) e CORBA (*Common Object Request Broker Architecture*, uma arquitetura para criação, distribuição e gerenciamento de objetos em rede proposta pelo *Object Management Group* – OMG -, um grupos de fornecedores de tecnologia de computação distribuída). (AYALA BROWNE et al, 2002)

As tecnologias tradicionais não resolveram os problemas da indústria, que buscava uma arquitetura consistente para interoperabilidade de softwares. A indústria procurava uma arquitetura que oferecesse neutralidade de fornecedor, plataforma, rede e linguagem. RMI e DCOM estavam altamente atrelados aos seus fornecedores, sua plataforma e linguagem de desenvolvimento. A alternativa, CORBA, utiliza um protocolo de comunicação que acaba por dificultar a interoperabilidade, além de ser uma tecnologia de complexa utilização.

Os *web services* surgem então como opção, oferecendo interoperabilidade baseada em tecnologias de acesso e utilização simplificada.

## 2.3 Benefícios

Os seguintes benefícios provocaram um aumento de interesse pela utilização de *web services*:

- Interoperabilidade: desenvolvedores de aplicações costumam criar seus sistemas em diferentes ambientes, linguagens e arquiteturas. Muitas vezes surge a necessidade de intercomunicação entre estas aplicações. Os *web services* oferecem interoperabilidade entre estes sistemas, independente da forma como foram desenvolvidos.
- Serviços de negócios através da Web: os *web services* permitem às empresas disponibilizar seus serviços através da web para qualquer cliente interessado, como, por exemplo, uma listagem de produtos.
- Integração de sistema existentes: muitas vezes, empresas possuem aplicações legadas onde o custo de remanejamento para uma nova tecnologia pode ser muito alto. *Web ser-*

*vices* permitem integrar sistemas legados e disponibilizar seus serviços já existentes para outros interessados sem a necessidade de remanejamento da aplicação.

- Liberdade de escolha: *web service* é um padrão que não é ligado a uma empresa fornecedora específica. Várias empresas desenvolvem soluções e ferramentas para facilitar o desenvolvimento de *web services*, permitindo aos clientes escolherem aquela que melhor se adequa a sua necessidade.
- Suporte a um maior número de clientes: *web services* podem ser acessados por variados tipos de clientes, desenvolvidos em diferentes linguagens.

## 2.4 Padrões e Tecnologias

Para permitir a interoperabilidade entre aplicações, deve-se ter ambientes respeitando padrões, de forma a facilitar a troca de informações entre as aplicações. Os *web services* empregam os seguintes padrões para alcançar este objetivo:

- Uma linguagem comum de comunicação – é necessário uma forma de comunicação padronizada para que o cliente e o servidor possam comunicar-se, uma linguagem que possa ser interpretada por ambos. Os *web services* utilizam para este propósito a *eXtensible Markup Language* (XML).
- Uma forma de troca de informações padrão – somente a definição de uma linguagem padrão não é o suficiente, é necessário também uma forma padrão para troca de informações. Os *web services* utilizam para isto o protocolo SOAP.
- Forma padrão para especificação de serviços – para prover serviços, deve haver um mecanismo padrão onde outras aplicações entendam o que está sendo oferecido. Para isto os *web services* utilizam o *Web Services Description Language* (WSDL).
- Forma padrão para encontrar serviços – da mesma forma que é necessário um mecanismo padrão para prover serviços, é preciso também um mecanismo padrão para encontrá-los. O *Universal Description, Discovery and Integration* (UDDI) é utilizado para este propósito.

### 2.4.1 XML

A linguagem XML (*eXtensible Markup Language* – linguagem de marcação extensível) é um padrão da W3C aceito pela indústria, independente de plataforma e tecnologias, baseado em um arquivo de texto simples, onde marcações (*tags*) são utilizadas para definir os contextos e seus valores. O XML foi definido como linguagem padrão para resolver o problema de se ter uma linguagem comum de comunicação entre as aplicações.

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ListaDeEnderecos>
  <Nome>Jose da Silva</Nome>
  <Endereco>
    <Rua>
      Rua Tenente Silveira, 159
    </Rua>
    <Cidade>
      Florianopolis
    </Cidade>
    <Estado>
      Santa Catarina
    </Estado>
    <Pais>
      Brasil
    </Pais>
  </Endereco>
  <Telefone>
    48 9999-9999
  </Telefone>
</ListaDeEnderecos>
```

**Figura 2.2: Exemplo de um arquivo XML**

Um documento XML pode estar associado a um documento DTD (*Document Type Definition* – definição de tipo de documento). Um DTD especifica quais marcações devem existir em um documento XML. Desta forma, em uma aplicação onde existe troca de arquivos XML, as partes sabem o que devem esperar no arquivo XML. Um XML pode ser submetido a um processo de validação de DTD, no qual é verificado se as regras descritas no DTD são respeitadas, fazendo com que o documento seja considerado válido.

```
<!ELEMENT ListaDeEnderecos(Nome,Endereco,Telefone) >
<!ELEMENT Nome (#PCDATA) >
<!ELEMENT Endereco (Rua,Cidade,Estado,Pais) >
<!ELEMENT Rua (#PCDATA) >
<!ELEMENT Cidade (#PCDATA) >
<!ELEMENT Estado (#PCDATA) >
<!ELEMENT Pais (#PCDATA) >
<!ELEMENT Telefone (#PCDATA) >
```

**Figura 2.3: Exemplo de um arquivo DTD**

Infelizmente, uma DTD é uma forma pobre de determinar a estrutura de um arquivo XML. Pensando nisso, a W3C, órgão que padroniza características da Web, desenvolveu uma forma alternativa de especificar o conteúdo de um documento XML, chamada de XML Schema.

Um XML Schema apresenta as seguintes características (W3 SCHOOLS, 2008a):

- Define que elementos podem aparecer no documento XML;
- Define atributos que um elemento pode possuir;
- Define quais são os elementos filho de um determinado elemento;
- Define a ordem em que os filhos podem ser determinados;
- Define o número de filhos de um elemento;
- Define quando um elemento pode estar vazio;
- Define os tipos de dados de um elemento;
- Define valores padrão para elementos;

Ainda, os XML Schemas também definem os chamados *Namespaces*. *Namespaces* foram criados para determinar o escopo de dois elementos com um mesmo nome. Por exemplo, em um documento mais complexo, onde temos "vendas" e "produtos", poderíamos ter duas *tags* chamadas "valor". Isto poderia causar confusão, pois não seria possível determinar se este valor refere-se a "vendas" ou a "produtos". O *Namespace* define a quem pertence este "valor".

A figura 2.4 apresenta um exemplo de um XML Schema:

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.inf.ufsc.br"
xmlns="http://www.inf.ufsc.br"
elementFormDefault="qualified">
  <xs:element name="ListaDeEnderecos">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Nome" type="xs:string"/>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Rua" type="xs:string"/>
              <xs:element name="Cidade" type="xs:string"/>
              <xs:element name="Estado" type="xs:string"/>
              <xs:element name="Pais" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
          <xs:element name="Telefone" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
  </xs:element>
</xs:schema>
```

**Figura 2.4: Exemplo de um XML Schema**

### 2.4.2 SOAP

O protocolo SOAP (acrônimo de *Simple Object Access Protocol* – protocolo de simples acesso a objetos – nome este que deixou de ser usado após a versão 1.2, pois SOAP deixou de ser uma abreviação (W3C, 2009)) foi especificado para suprir a necessidade de uma forma de troca de mensagens comum entre aplicações. SOAP permite que aplicações que não se conheçam troquem informações sobre seus objetos. SOAP é baseado em arquivos XML, o que o torna independente de plataformas e arquiteturas.

SOAP funciona como um envelope sobre um objeto XML, especificando informações sobre o serviço a ser invocado, codificação (*encoding*) da resposta, dentre outros. No corpo de uma mensagem SOAP podem ser enviados parâmetros solicitados pelo serviço, ou a própria resposta de um serviço.

O SOAP em si é também um documento XML, apenas adicionando informações relativas a troca das mensagens. A figura 2.5 demonstra a arquitetura de um documento SOAP:

```xml
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

**Figura 2.5: Envelope SOAP**

Um documento SOAP contém um cabeçalho (*env:Header*) e um corpo (*env:Body*). No cabeçalho podem ser definidas informações úteis aos intermediários SOAP, como no exemplo acima, onde um controle (*n:alertcontrol*) informa a prioridade (*n:priority*) da mensagem e sua data de expiração (*n:expires*). No corpo é definida a mensagem propriamente dita.

### 2.4.3 WSDL

O WSDL (*Web Services Description Language* – Linguagem de descrição de serviços web) provê uma forma padrão para especificação de serviços. O WSDL é um documento XML que especifica os detalhes de cada serviço: suas interfaces de comunicação, parâmetros, respostas, descrição e detalhes de seu funcionamento. Desta forma, um cliente com interesse no serviço pode conhecer todas as suas características acessando seu WSDL (W3, 2009).

Um documento WSDL é composto pelas seguintes partes:

- *Types:* definem os tipos dos dados contidos em mensagens trocadas pelo serviço. Podem ser simples, complexos, derivados ou um vetor de dados;
- *Messages:* definem as mensagens trocadas pelo serviço;
- *Port Types:* definem as operações que são disponibilizadas por um *web service*;
- *Bindings:* definem como os *port types* funcionam em conjunto com as mensagens (*Messages)*;
- *Services:* os serviços e os *port types* juntos definem um *web service*, provendo um endereço único para acessá-lo;

A figura 2.6 demonstra um exemplo de um documento WSDL, no qual é especificado um serviço para obtenção de endereços de empresas cadastradas em uma base de dados.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="EnderecosWebService"
  xmlns:tns="urn:EnderecosWebService"
  xmlns="http://schemas.xmlsoap.org/wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap" >

  <types/>

  <message name="EnderecosService_getEndereco">
    <part name="String_1" type="xsd:string" />
  </message>
  <message name="EnderecosService_getEnderecoResponse">
    <part name="result" type="xsd:string" />
  </message>

  <portType name="EnderecosService">
    <operation name="getEndereco" parameterOrder="String_1">
      <input message="tns:EnderecosService_getEndereco" />
      <output message="tns:EnderecosService_getEnderecoResponse" />
    </operation>
  </portType>

  <binding name="EnderecosServiceBinding" type="tns:EnderecosService">
    <operation name="getEndereco">
      <input>
        <soap:body use="literal"
          namespace="urn:EnderecosWebService" />
      </input>
      <output>
        <soap:body use="literal"
          namespace="urn:EnderecosWebService" />
      <output>
        <soap:body use="literal"
          namespace="urn:EnderecosWebService" />
      </output>
      <soap:operation soapAction=""/>
    </operation>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="rpc" />
  </binding>

  <service name="EnderecosWebService">
    <port name="EnderecosServicePort" binding="tns:EnderecosServiceBinding" >
      <soap:address location="http://minhaempresa.com/enderecosservice" />
    </port>
  </service>

</definitions>
```

**Figura 2.6: Exemplo de um WSDL**

### 2.4.4 UDDI

O UDDI (*Universal Description, Discovery and Integration* – descrição, descoberta e integração universais) é uma especificação que determina como serviços podem ser registrados, cancelados e encontrados. A ideia é funcionar como uma lista telefônica, onde serviços são listados juntamente com a forma de utilizá-los (no caso, o número de um telefone).

A especificação UDDI determina duas APIs (*Application Programming Interface* – interface para programação de aplicações) de acesso:

- API de busca – utilizada por clientes que desejam pesquisar serviços disponíveis. Com ela o cliente tem acesso a informações básicas do serviço, como seu nome e sua descrição, além de especificações técnicas de utilização, como os dados de seu WSDL (parâmetros, funcionamento, etc.).
- API de registro – utilizada por aplicações que desejam se registrar na listagem de serviços. A aplicação deve fornecer os seus dados técnicos para se inscrever no serviço de busca.

O UDDI em si funciona como um *web service*, utilizando também SOAP e XML para a troca de informações.

# 3 BLUETOOTH

Bluetooth é uma tecnologia de comunicação sem fio a curta distância, que visa substituir a utilização de cabos para a comunicação entre dispositivos eletrônicos através da aplicação de uma arquitetura WPAN. Outro exemplo de tecnologia que utiliza a arquitetura WPAN é o Zigbee (Zigbee, 2009).

Desenvolvida inicialmente pela companhia sueca Ericsson (*Telefonaktiebolaget L. M. Ericsson*) durante o ano de 1994, surgiu da necessidade de fazer com que computadores portáteis se comunicassem com celulares. Após sua proposição inicial, a Ericsson convidou outras grandes companhias para se unirem e formular uma especificação.

Desta união surgiu, em 1998, o *Bluetooth Special Interest Group* (BLUETOOTH SIG, 2008b), uma associação que reúne empresas líderes em telecomunicações, computação, mobilidade, automação industrial e redes. O SIG tem a função de especificar as diretrizes da tecnologia Bluetooth, administrar um programa de qualificação, proteger a marca Bluetooth e difundir a utilização da tecnologia. Dentre as grandes empresas que fazem parte do grupo estão a Ericsson, Intel, Lenovo, Microsoft, Motorola, Nokia e Toshiba, além de várias outras companhias, que atualmente constituem cerca de 9000 empresas.

O nome surgiu por parte de um funcionário da Intel (EETIMES, 2008) que ficou sabendo de uma história sobre um rei escandinavo do século 10, Harald Bluetooth (cujo nome original era Harald Blaatand). Segundo a história, Harald foi responsável por unificar os reinos escandinavos, assim como o SIG agora tentava unificar os fabricantes em torno de uma tecnologia de interesse comum.

## 3.1 Desafios

A tecnologia Bluetooth foi baseada nas seguintes premissas especificadas pelo Bluetooth SIG:

- Suporte a dados e voz: a tecnologia deve ser capaz do prover transmissão de dados e voz com boa qualidade, sendo considerada de boa qualidade a atual transmissão de voz por telefones com fio.
- Permitir conexões *ad-hoc*: a natureza dinâmica dos dispositivos móveis torna complexa a tarefa de assumir qualquer informação referente ao ambiente de operação. Assim sendo, a tecnologia deve poder detectar e estabelecer conexões com qualquer outra unidade compatível.
- Poder lidar com interferência causada por outras fontes: o Bluetooth opera em ondas de rádio na frequência de 2.4GHz, que é utilizada por vários outros dispositivos, por ser uma frequência de rádio não licenciada. A tecnologia deverá saber lidar com possíveis interferências causadas por estes dispositivos.
- Utilização mundial: a tecnologia deve se adequar às diferenciações relativas à variação de rádio frequência que ocorre em vários países.
- Proteção similar à de uma transmissão de um cabo: a tecnologia deve prover mecanismos de segurança e autenticação, visto que seus usuários não desejarão que seus dados possam ser interceptados por outros receptores diferentes do escolhido pelo usuário.
- Tamanho pequeno: o módulo de rádio deve ser pequeno o bastante para poder ser utilizado em vários dispositivos portáteis.
- Baixo consumo de energia: vários dos dispositivos que utilizarão a tecnologia serão movidos a bateria. Esta característica implica que a tecnologia não deverá comprometer o tempo de vida das baterias dos dispositivos.

## 3.2 Arquitetura

A arquitetura Bluetooth é formada por um sistema físico de transmissão de ondas de rádio e por uma pilha de software responsável por interligar as sub-camadas definidas pela especificação Bluetooth.

A subdivisão da camada de software visa facilitar a interoperabilidade entre dispositivos e facilitar a adoção do Bluetooth como forma de comunicação.

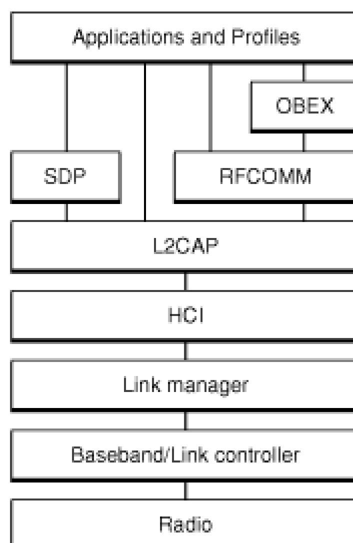



**Figura 3.1: A pilha de protocolo Bluetooth (APPLE2, 2008b)**

### 3.2.1 Camada inferior

Na base da pilha de protocolo, conforme demonstrado na figura anterior, está a camada de rádio, que é responsável pela modulação e demodulação de dados em sinais de rádio-frequência. A camada de rádio deve ser capaz de transmitir e receber sinais de rádio frequência nas faixas determinadas pela especificação Bluetooth (2.4 Ghz, uma banda de frequência que pode ser utilizada livremente para aplicações industriais, científicas e médicas, definida pela ITU – *International Comunication Union* (INTERNATIONAL TELECOMMUNICATION, 2008)).

Um nível acima se encontra a camada *Baseband*, responsável por realizar a conversão dos dados em frequência de rádio para as camadas lógicas superiores, e vice-versa.

A camada *Link Manager Protocol* é responsável por controlar e negociar todos os aspectos de conexão entre dois dispositivos, estando incluídos os processos de configuração e controle de conexões lógicas e físicas, controle de desconexões, autenticação, criptografia e sincronização entre os dispositivos.

Por fim, temos a camada HCI (*Host Controller Interface*), que atua como fronteira entre a camada inferior e a camada superior da pilha de protocolo, realizando a troca de informações entre as duas camadas.

### 3.2.2 Camada Superior

Acima da camada HCI se encontra a camada superior da pilha de protocolo Bluetooth. A primeira delas é a L2CAP (*Logical Link Control and Adaptation Protocol*). Suas principais responsabilidades são:

- Estabelecer conexões sobre links de comunicação entre dispositivos já existentes, ou solicitar um novo link caso ele ainda não exista.
- Preparar os dados para a forma esperada pela camada solicitante.

A camada L2CAP utiliza o conceito de canais para saber de onde vem e para onde devem ir os pacotes de dados transitantes. Um canal é uma representação lógica de um fluxo de comunicação entre as camadas L2CAP de dois dispositivos diferentes.

Acima da L2CAP, temos as camadas lógicas que visam oferecer serviços de uso geral e facilitar o desenvolvimento de aplicações Bluetooth.

A camada SDP – *Service Discovery Protocol* – define uma série de ações para procura de serviços disponíveis. Segundo a especificação Bluetooth, um serviço é qualquer atividade disponibilizada por outro dispositivo Bluetooth remoto. Dispositivos Bluetooth podem tanto ser clientes ou servidores de serviços Bluetooth. Um cliente da camada SDP comunica-se com um servidor SDP de outro dispositivo através de um canal reservado pela L2CAP, e recebe uma lista de serviços disponíveis. Sabendo quais são os serviços disponíveis, pode-se então solicitar um link de comunicação com o serviço para iniciar-se a troca de informação.

A camada RFCOMM emula uma troca de dados serial, como acontece em uma comunicação por cabos seriais. Assim, aplicações legadas que utilizem este tipo de comunicação podem ser migradas para utilizar comunicação Bluetooth.

A camada de comunicação OBEX oferece suporte à troca de dados seguindo o protocolo OBEX. OBEX é um protocolo de transferência que visa simplificar a troca de dados como objetos entre dispositivos de forma rápida e concisa, oferecendo suporte a trocas sim-

ples de objetos entre dispositivos e também a troca de objetos durante sessões, mantendo a conexão ativa mesmo enquanto não está sendo utilizada.

### 3.2.3 Perfis Bluetooth

Perfis Bluetooth definem os possíveis tipos de aplicações Bluetooth. Os perfis definem comportamento genéricos esperados durante uma comunicação Bluetooth entre dispositivos.

Exemplos de perfis existentes (BLUETOOTH SIG, 2009):

- *Basic Printing Profile* (BPP): permite que dispositivos enviem textos, emails, imagens ou outros itens para serem impressos;
- *File Transfer Profile* (FTP): define como arquivos e pastas de um servidor podem ser acessados por uma aplicação cliente;
- *Headset profile* (HSP): define como deve ocorrer a troca de informações entre dispositivos do tipo *headset* (fone de ouvido com microfone acoplado);
- *Human Interface Device Profile* (HIDP): define protocolos, procedimentos e capacidades de dispositivos de interação humana, como teclados, mouses e dispositivos de controle de jogos;

## 3.3 Topologia de comunicação

*Links* de comunicação Bluetooth são formados sempre no contexto de uma *piconet*. Uma *piconet* é constituída de dois ou mais dispositivos compartilhando um mesmo canal de comunicação. Neste canal, os dispositivos são sincronizados a partir do *clock* de um e somente um dispositivo, conhecido como *master* (mestre) da *piconet*. Todos os outros dispositivos são sincronizados a partir do *master*, e são conhecidos como *slaves* (escravos) da *piconet*.

Em um ambiente Bluetooth, pode-se ter várias *piconets* simultaneamente. Um mesmo dispositivo pode participar de várias das *piconets*. No entanto, um dispositivo só pode ser o mestre de apenas uma *piconet*, isto porque o dispositivo mestre é responsável por sincronizar a *piconet*, e não é possível um mesmo dispositivo sincronizar mais de uma *piconet* simulta-

neamente. Porém, como dispositivo escravo, o dispositivo pode participar de várias *piconets*. Neste caso, diz-se que o dispositivo faz parte de uma *scatternet*.

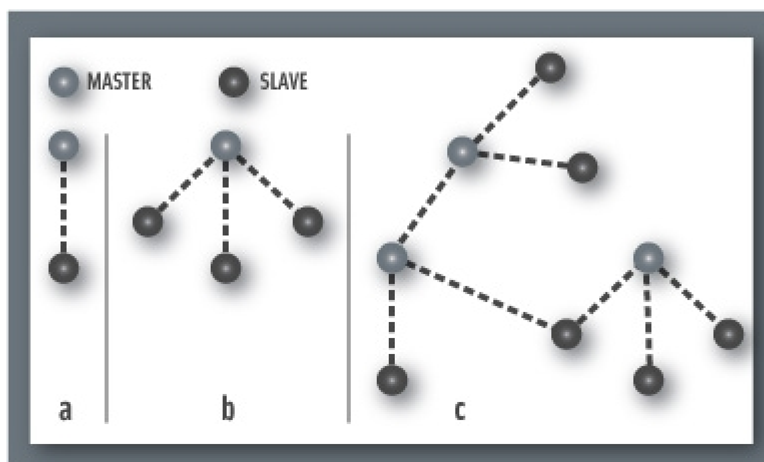



**Figura 3.2: Redes Bluetooth**

### 3.3.1 Operações de comunicação

A tecnologia Bluetooth funciona como uma rede sem fio *ad-hoc*, onde dispositivos em um raio de alcance podem descobrir outros dispositivos, e assim formar um canal de comunicação. Para isso, a especificação Bluetooth estabelece um processo a partir do qual um canal de comunicação entre dispositivos é estabelecido.

A primeira etapa é conhecida como *Inquiry*, ou processo de descoberta. Dispositivos Bluetooth utilizam o processo de descoberta para encontrarem outros dispositivos na área, ou para serem descobertos por outros dispositivos. O processo de descoberta ocorre de forma assímetrica. Um dispositivo realizando o *Inquiry* envia sinais solicitando que outros dispositivos na área se identifiquem. Posteriormente, dispositivos que possam atender ao chamado respondem avisando que estão disponíveis.

Em uma segunda etapa, os dispositivos entrarão no processo de conexão (*Connecting Procedure*). Esta etapa também ocorre de forma assimétrica. Um dispositivo fica responsável por estabelecer a conexão, enquanto o outro se torna um dispositivo conectável. A conexão é

solicitada pelo primeiro dispositivo, através de um canal de comunicação específico do Bluetooth especialmente designado para tratar conexões entre os dispositivos. Os dispositivos conectáveis respondem à solicitação de conexão, e um canal de comunicação é então formado, ficando estabelecida uma rede *piconet*.

Finalmente, os dispositivos estão habilitados a realizarem troca de mensagens através da rede *piconet*.

# 4 JAVA

Java refere-se a um conjunto de tecnologias de desenvolvimento de software, formado pela linguagem de programação Java, a máquina virtual Java e a plataforma Java.

A tecnologia Java surgiu como evolução de um projeto interno da Sun Microsystems, o *The Green Project* (O Projeto Verde) (SUN, 2008a). Em 1991, a Sun reuniu um grupo de 13 pessoas, e definiu como missão do grupo antecipar qual seria a nova onda da computação. O grupo chegou à conclusão de que uma das novas ondas seria a inclusão cada vez maior de componentes computacionais em dispositivos que fazem parte do dia a dia das pessoas.

Com esta ideia em mente, a equipe fez uma demonstração de um protótipo, que demorou 18 meses para ser desenvolvido. O protótipo era um dispositivo de mão, com uma tela sensível ao toque, onde várias animações e aplicações podiam ser executadas. A razão pela qual o dispositivo conseguia executar várias aplicações diferenciadas era que ele estava sendo executado sob uma nova linguagem, que era independente de processador.

**Figura 4.1: O protótipo do Green Project (SUN, 2008b)**

Neste protótipo foi também a primeira vez onde apareceu o atual mascote da tecnologia, conhecido como Duke:

**Figura 4.2: Duke, o mascote (JAVANET, 2008)**

O desenvolvimento de dispositivos com conteúdo digital ainda estava imaturo na época, e a tecnologia acabou não ficando popular. Porém, uma outra onda estava iniciando sua expansão: a Internet.

Java acabou por ser uma das primeiras tecnologias a oferecer conteúdo visual dinâmico em navegadores web, através dos *Java Applets*. Com o passar do tempo, a tecnologia cresceu, evoluiu para outros ramos, e hoje é uma das plataformas mais populares para desenvolvimento em várias frentes.

## 4.1 A tecnologia

Java refere-se basicamente a três frentes de tecnologia: a linguagem de programação Java, a máquina virtual Java e a plataforma de desenvolvimento Java. A linguagem de programação Java tem sua sintaxe baseada principalmente em C++. É uma linguagem de programação orientada a objetos, utilizando os conceitos de classes (definições das características de alguma coisa, como suas propriedades e seus comportamentos), objetos (um exemplar de uma classe), métodos (comportamentos), herança (versões especializadas de classes herdam as características de suas classes pais) e polimorfismo (um objeto pode ser tratado como de outra classe para que realize um comportamento diferente de seu comportamento original).

A máquina virtual Java é onde os códigos-fontes Java compilados são executados. Uma das ideias originais da plataforma Java era que ela pudesse ser executada independentemente de processador. Para atingir este objetivo, foi definido que um código-fonte Java, diferentemente das compilações tradicionais, onde a compilação é feita diretamente para um código que será executado por uma máquina, seria compilado para uma linguagem intermediá-

ria. O código intermediário é chamado de *bytecode*. Este código intermediário será interpretado por máquinas virtuais, desenvolvidas para interpretar os *bytecodes* para várias arquiteturas de processadores. Essas são as chamadas máquinas virtuais Java.

Java cresceu para atender demandas de várias frentes. A partir deste crescimento, a tecnologia foi dividida em três plataformas principais. Uma plataforma é um conjunto formado pela linguagem de programação, um grupo de bibliotecas padrão e uma máquina virtual para executar o código. Atualmente, a plataforma Java possui três edições de desenvolvimento: JSE, JEE e JME.

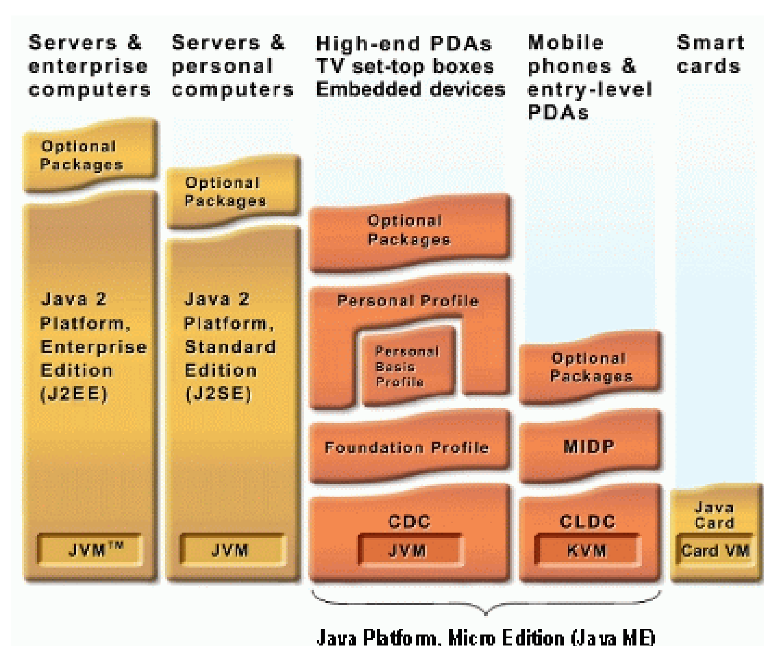


**Figura 4.3: As edições da plataforma Java**

JSE (*Java Standard Edition*) é a versão padrão para computadores pessoais. A plataforma JSE provê um conjunto de ferramentas para desenvolvimento de aplicações para desktop, com suporte ao desenvolvimento de GUIs (*Graphical User Interface* - Interface Gráfica para Usuários), acesso a bancos de dados, e, atualmente, suporte a execução de linguagens de scripting, como Ruby e Python (SUN, 2008c).

JEE (*Java Enterprise Edition*) é uma extensão da plataforma JSE, e fornece suporte ao desenvolvimento de aplicações corporativas que podem ser acessadas através da Web (SUN, 2008d). Essa versão do Java permite desacoplar aplicações de suas camadas de negócios e fornece mecanismos para construção de aplicações utilizando a arquitetura SOA (*Ser-*

*vice Oriented Architecture* - Arquitetura Orientada a Serviços), na qual atividades de negócio são encapsuladas e fornecidas como um serviço a consumidores (OPEN GROUP, 2008a).

Por fim, JME (*Java Micro Edition*) provê um ambiente para execução de aplicações em dispositivos móveis, como celulares, PDAs (*personal digital assistants* – assistentes pessoais digitais) e televisores. Provê suporte ao desenvolvimento de interfaces gráficas para estes dispositivos, além de configurações de rede e comunicação (SUN, 2008e).

## 4.2 Java APIs for Bluetooth

Para utilização da tecnologia Bluetooth juntamente com a plataforma Java, foi criada a especificação JSR 82: *Java APIs for Bluetooth*, com a finalidade de padronizar o desenvolvimento de aplicações Bluetooth para a plataforma. A especificação visa remover a complexidade de se lidar com a pilha de protocolos Bluetooth, e permitir que desenvolvedores foquem no desenvolvimento de aplicações. A especificação é mantida pelo JCP (*Java Community Process*), um grupo com o objetivo de desenvolver e revisar especificações relativas à plataforma Java (JAVA COMMUNITY PROCESS, 2008).

A especificação define uma API de acesso e controle a dispositivos Bluetooth. Esta API deve ser implementada pelos desenvolvedores de dispositivos Bluetooth que queiram aderir à especificação. Desta forma, desenvolvedores de aplicativos podem criar suas aplicações da mesma forma, independentemente do dispositivo sendo utilizado. Ainda, a especificação provê suporte básico aos protocolos e perfis Bluetooth, e não inclui APIs específicas para os perfis porque o número de perfis é grande e tende a crescer ainda mais.

A API visa prover as seguintes capacidades (SUN DEVELOPERS, 2008):

- Registrar serviços;
- Encontrar dispositivos e serviços;
- Estabelecer comunicação utilizando RFCOMM, L2CAP ou OBEX;
- Utilizar as conexões para enviar e receber mensagens;
- Gerenciar as comunicações entre dispositivos;
- Prover segurança na comunicação entre os dispositivos;

A API é dividida em dois pacotes principais: *javax.bluetooth* e *javax.obex*. A API para utilização de OBEX é separada pois pode ser utilizada também em outras tecnologias.

### 4.2.1 Utilização

Na especificação, dispositivos podem existir de duas maneiras: como um *LocalDevice* ou como um *RemoteDevice*. Um *LocalDevice* representa um dispositivo local, o próprio dispositivo onde se está executando o código. Um *RemoteDevice* representa um dispositivo remoto, ao qual se deseja realizar uma comunicação a partir do dispositivo local.

Como os dispositivos que utilizam a tecnologia Bluetooth são primariamente de natureza móvel, deve haver uma forma de encontrar, dentro de uma determinada área, quais são os dispositivos disponíveis. Na API, a busca de dispositivos é feita através da classe *DiscoveryAgent*, e pode procurar dispositivos de duas formas: pesquisar novos dispositivos disponíveis na área, ou pesquisar por dispositivos que já foram encontrados em consultas anteriores.

A mesma classe *DiscoveryAgent* também é responsável por encontrar serviços disponibilizados pelos dispositivos encontrados. Primeiramente, um serviço deve ter sido disponibilizado para acesso por um dispositivo. Este processo é chamado de *service registration* – registro de serviço. No processo de registro, um serviço deve receber um UUID (*Universally Unique Identifier* – Identificado único universal) a partir do qual outros dispositivos irão poder encontrá-lo. Ainda no processo de registro, configurações relativas a segurança, como criptografia dos dados, são determinadas.

Com serviços disponibilizados na área, outros dispositivos podem encontrá-los, e então estabelecer um canal de comunicação entre ambos. Serviços são pesquisados pelos seus identificadores universais (UUIDs). Todos os dispositivos pesquisados na área que contiverem serviços com os UUIDs solicitados pelo pesquisador serão retornados.

O dispositivo que está efetuando a busca recebe uma lista de URLs (*Uniform Resource Locator* – Localizador uniforme de recursos) com os endereços relativos aos serviços encontrados. A partir desta URL, os dois dispositivos criam um canal de comunicação através do qual podem realizar o envio e recebimento de mensagens do serviço.

# 5 INTEGRAÇÃO DE DISPOSITIVOS COM WEB SERVICES

A computação ubíqua tem recebido um forte foco de estudo atualmente. Temos acompanhado a evolução dos sistemas computacionais de forma que o modelo de computação centralizada tem sido substituído por um modelo de computação distribuída. Neste contexto, dispositivos de diferentes categorias, desde computadores interligados por meio de redes tradicionais até dispositivos móveis como celulares, sensores móveis e tags RFID, podem fazer parte de um mesmo ambiente.

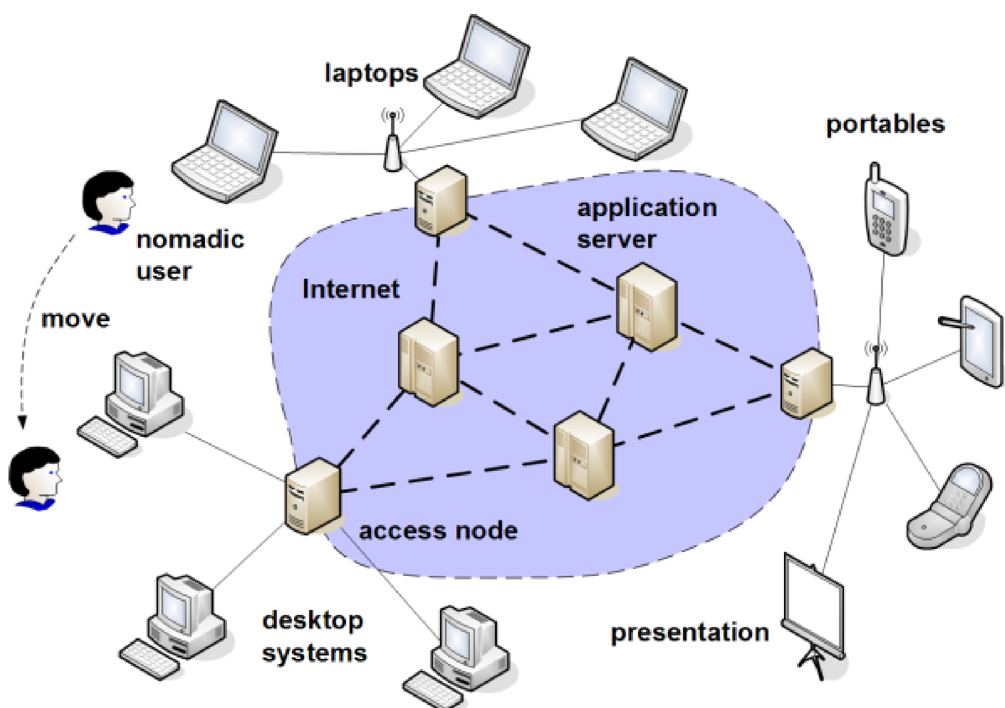



**Figura 5.1: Diferentes dispositivos em conjunto**

A figura 5.1 apresenta um exemplo de um ambiente ubíquo, com dispositivos de diferentes categorias agindo em conjunto para fornecer informações a usuários e a outros dispositivos.

Diferentes categorias de dispositivos móveis e tecnologias de comunicação implicam em diferentes padrões de redes e de comunicação, e a integração destes diferentes padrões é uma área de estudos ainda em aberto.

Estes novos dispositivos que hoje fazem parte de nosso dia-a-dia têm recebido novas características, como aumento de poder de processamento e funcionalidades de comunicação com redes *wireless*, permitindo a comunicação com outros dispositivos, ou até mesmo executar protocolos de comunicação como os requeridos pela arquitetura SOA. Apesar do aumento das capacidades destes dispositivos, a falta de padronização tem dificultado a intercomunicação entre eles. Em meio a isto, uma das propostas de intercomunicação entre dispositivos é o DPWS, um padrão baseado em serviços Web, proposto por um grupo de empresas, que visa permitir a interoperabilidade entre dispositivos, não limitando-se a determinadas características de rede ou de plataforma computacional.

## 5.1 DPWS – Devices Profile for Web Services

O DPWS é um padrão de comunicação para dispositivos de recursos restritos, baseado no padrão de disponibilização de serviços proposto pelos serviços web. O DPWS organiza um conjunto de mecanismos empregados pelos serviços web de forma a adequá-los à utilização por dispositivos de recursos restritos, e é uma especificação para aqueles que estão desenvolvendo bibliotecas de comunicação de dispositivos em rede (MSDN, 2008a).

Para uma comunicação básica entre dispositivos seguindo o padrão de web services, o DPWS descreve como as funcionalidades padrões devem acontecer para se prover as seguintes funcionalidades de comunicação:

- Enviar e receber mensagens de serviços web;
- Descobrir dinamicamente serviços web;
- Descrever serviços web;
- Inscrever-se e receber notificações de eventos de serviços web.

A figura 5.2 apresenta um diagrama de sequência com a comunicação proposta pelo DPWS.

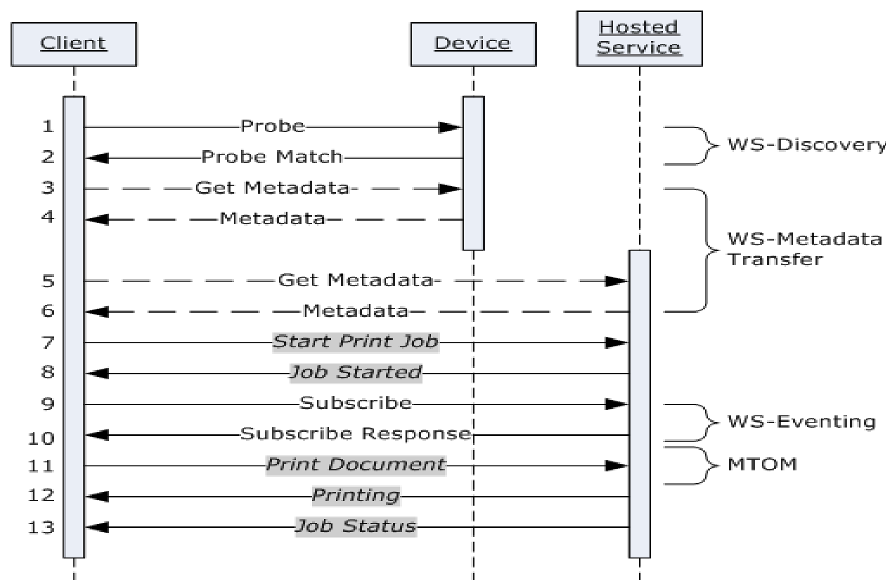



**Figura 5.2: Comunicação proposta pelo DPWS (MSDN, 2008b)**

Os padrões adicionais relacionados a *web services* utilizados pelo DPWS são os seguintes (ZEEB BOBEK et al, 2007):

- *WS-Addressing*: Efetua o endereçamento de serviços web. Tradicionalmente, o protocolo SOAP é utilizado juntamente com o transporte via HTTP. Esta conjunção trouxe, por definição, algumas restrições ao uso de serviços web, como a troca de mensagens de forma síncrona. A extensão introduz o conceito de *endpoint references*, onde os serviços web possuem um endereço único (uma URI), e os *message information headers*, que contém informações sobre a mensagem transmitida. Cada mensagem pode ter um identificador único, podendo assim uma mensagem referenciar uma outra, permitindo assim trocas de mensagens de forma assíncrona;
- *WS-Metadata Exchange:* Troca de meta-dados de serviços web. Define operações que provêem informações de meta-dados sobre os serviços, como formas de comunicação, tipos de dados suportados e operações disponíveis;
- *WS-Discovery:* Descoberta de serviços web. Provê mecanismos de busca de dispositivos e seus serviços em uma rede;
- *WS-Policy:* Políticas de serviços web. Diferentes serviços podem possuir diferentes características, requisitos e capacidades. O *framework* de políticas permite que este tipo de declaração possa ser descrito para que clientes compreendam a estrutura do serviço oferecido;

- *WS-Eventing:* Eventos de serviços web. Define um conjunto de operações que permitem a um cliente se inscrever em um serviço web, de forma que possa receber uma notificação em um outro momento, tornando possível a troca de mensagens de forma assíncrona;
- *WS-Security:* Segurança de serviços web. Provê funções de segurança para trocas de mensagens, com o intuito de garantir a autenticidade, a confidencialidade e a integridade dos dados;

## 5.2 DSB – Device Service Bus

A arquitetura proposta pelo DPWS é robusta e trata vários aspectos referentes à utilização de serviços web por dispositivos. Porém, nem todo dispositivo tem a capacidade de executar uma implementação da arquitetura, principalmente em se tratando de dispositivos de recursos limitados, como celulares e leitores RFID. Em todo caso, muitas vezes estes dispositivos possuem alguma tecnologia de comunicação como, por exemplo, RFID e Bluetooth.

O *Device Service Bus* é uma camada de infra-estrutura baseada em uma plataforma leve e portável, que pode ser executada em estações de trabalho e pequenos dispositivos. Seguindo os princípios do SOA, o DSB provê um meio de integrar dispositivos, agindo como um intermediário entre provedores e consumidores de serviços (ARAUJO SIQUEIRA, 2008).

Desta forma, o principal objetivo do DSB é integrar dispositivos que utilizem tecnologias específicas de comunicação e prover seus serviços como serviços web. A figura 5.3 demonstra a arquitetura proposta pelo DSB.

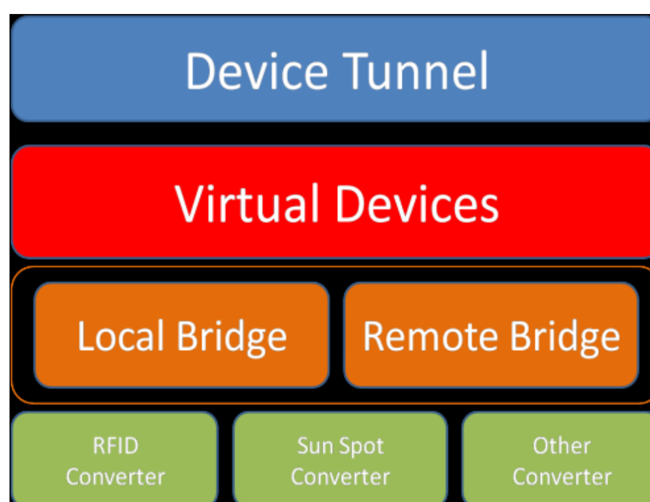



**Figura 5.3: A arquitetura DSB**

O *Device Tunnel* é uma implementação de uma pilha DPWS, e expõe dispositivos e serviços encontrados nas *Bridges* e nos *Converters.*

Um *Virtual Device* representa dispositivos que não implementam naturalmente uma pilha DPWS. Um *Virtual Device* pode conter vários *Virtual Services* e *Virtual Actions,* que representam os serviços disponibilizados pelo dispositivo.

As *bridges* mantém um cache de *Virtual Devices,* e controlam a entrada e saída de dispositivos e serviços que são encontrados. Também, são responsáveis pelas interações entre dispositivos virtuais e os dispositivos reais. Uma *bridge* pode ser local ou remota. Uma *bridge* local é executada no próprio dispositivo que oferecerá os serviços. A *bridge* remota pode ser utilizada quando o dispositivo não tem capacidade de executar ele mesmo uma pilha DSB.

Por fim, os *Converters* implementam a comunicação com tecnologias específicas, tornando-as disponíveis para as camadas superiores. Desta forma, dispositivos que não implementam uma pilha DPWS podem fazer parte de um sistema DPWS. Os *Converters* conhecem as características específicas das tecnologias envolvidas, e devem saber extrair informações relativas aos dispositivos e aos serviços oferecidos.

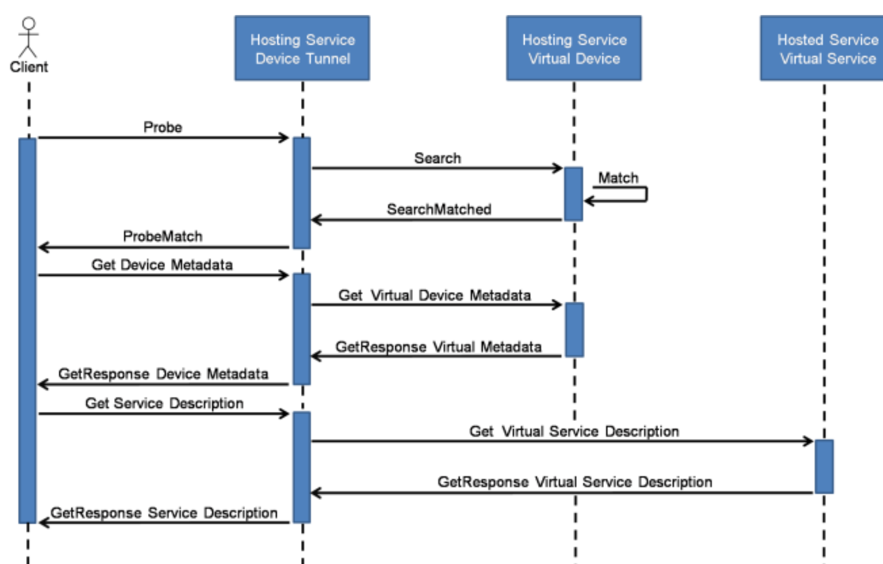



**Figura 5.4: Dinâmica de busca de dispositivos e serviços**

A figura 5.4 demonstra o fluxo de comunicação em uma busca por dispositivos e serviços. Primeiro, um cliente faz uma busca por uma determinada característica. O DSB retorna os dispositivos/serviços que atendam a solicitação, e o cliente pode obter informações de meta-dados relativas ao dispositivo e ao serviço.

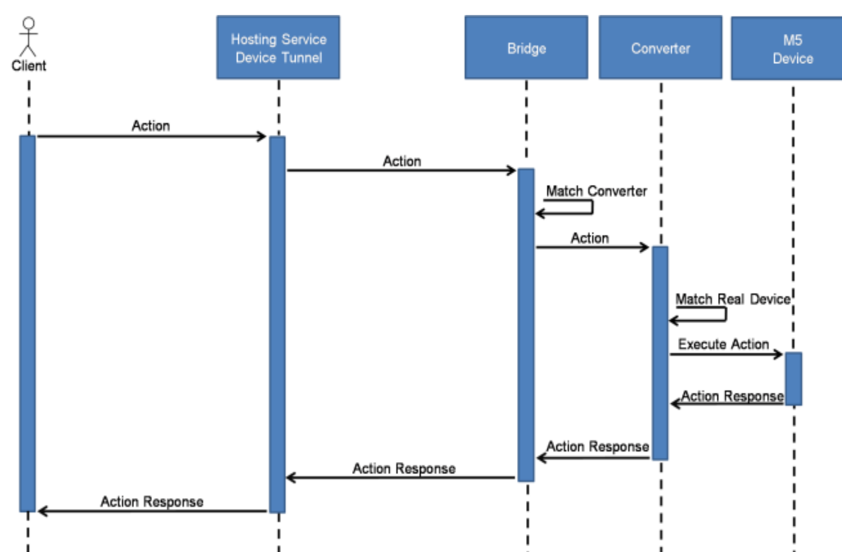



**Figura 5.5: Dinâmica de invocação de um serviço**

Para a execução de um serviço, um cliente faz a solicitação, seguindo o procedimento mostrado na figura 5.5. A *bridge* correspondente encaminha a solicitação ao *Converter* responsável pela tecnologia envolvida. O *Converter* deve então ser capaz de identificar o dispositivo referente ao serviço solicitado, encaminhar a requisição, aguardar a resposta do dispositivo e devolver uma resposta à *bridge*, que então devolverá a resposta para o cliente.

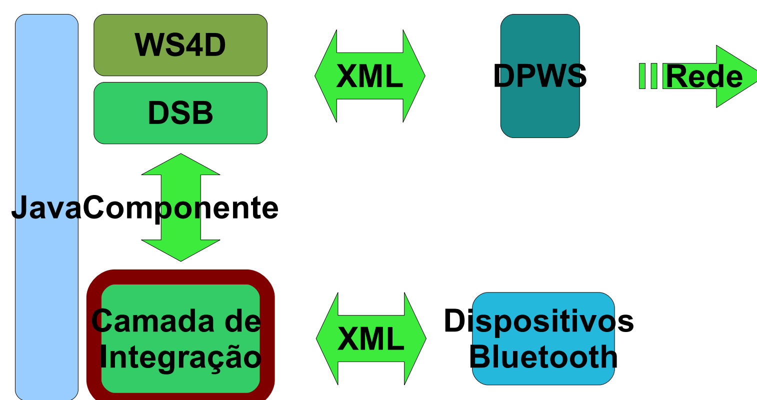



**Figura 5.6: Arquitetura dos componentes envolvidos**

A figura 5.6 apresenta todos os componentes envolvidos na integração. A tecnologia Java é a base para o desenvolvimento. O DSB utiliza a implementação WS4D do DPWS. O *BluetoothConverter* é um componente do DSB, e realiza a integração com dispositivos Bluetooth.

# 6 INTEGRAÇÃO DA TECNOLOGIA BLUETOOTH AO DSB

O objetivo final deste trabalho é desenvolver um *Converter* que permita que dispositivos com a tecnologia Bluetooth possam fazer parte da arquitetura DSB. Para isto, foi desenvolvido o *BluetoothConverter*, um *Converter* seguindo as características do DSB que será responsável por interagir com a tecnologia Bluetooth.

## 6.1 Diagrama de classes

A figura 6.1 apresenta o diagrama de classes das entidades envolvidas no desenvolvimento do *BluetoothConverter*. As classes estão organizadas em pacotes, cada qual representando uma etapa do *BluetoothConverter*. Os seguintes pacotes fazem parte do *BluetoothConverter*:

- *br.ufsc.inf.ppgcc.converter.bluetooth* – contém a classe *BluetoothConverter* e a interface de comunicação com a camada Bluetooth *BluetoothHandler*;
- *br.ufsc.inf.ppgcc.converter.bluetooth.impl* – contém a implementação do *BluetoothHandler* com o *framework* Marge[2], *BluetoothHandlerMargeImpl*, e a classe utilitária *BluetoothHandlerUtils*;
- *br.ufsc.inf.ppgcc.converter.bluetooth.impl.communication* – contém a fábrica de comunicação *ActionConverterCommunicationListenerFactory* e a entidade de comunicação *ActionConverterCommunicationListener;*
- *br.ufsc.inf.ppgcc.converter.bluetooth.services* – contém a interface do serviço de busca e invocação DSB para clientes Bluetooth, *BluetoothDSBServicesProvider;*

---

**2** O *framework Marge será descrito na seção 6.2.1*

- *br.ufsc.inf.ppgcc.converter.bluetooth.services.impl* – contém a implementação da interface de serviço com o *framework* Marge, *BluetoothDSBServicesProviderImpl;*

- *br.ufsc.inf.ppgcc.converter.bluetooth.services.messages* – contém as entidades que encapsulam as mensagem trocadas entre o provedor de serviços DSB e clientes Bluetooth, *BluetoothTunnelSearchRequest* e *BluetoothTunnelSearchResponse*;

- *br.ufsc.inf.bluetooth.services* – contém serviços com a finalidade de exemplificar o uso do *BluetoothConverter*. *StackEmulator* inicia o emulador de uma pilha Bluetooth utilizando a biblioteca Bluecove. *EchoBluetoothService* é um serviço Bluetooth para utilizar de exemplo no *BluetoothConverter*. *BluetoothDiscoveryServiceConsumer* é um exemplo de como consumir o serviço de busca e invocação do DSB com um cliente Bluetooth.

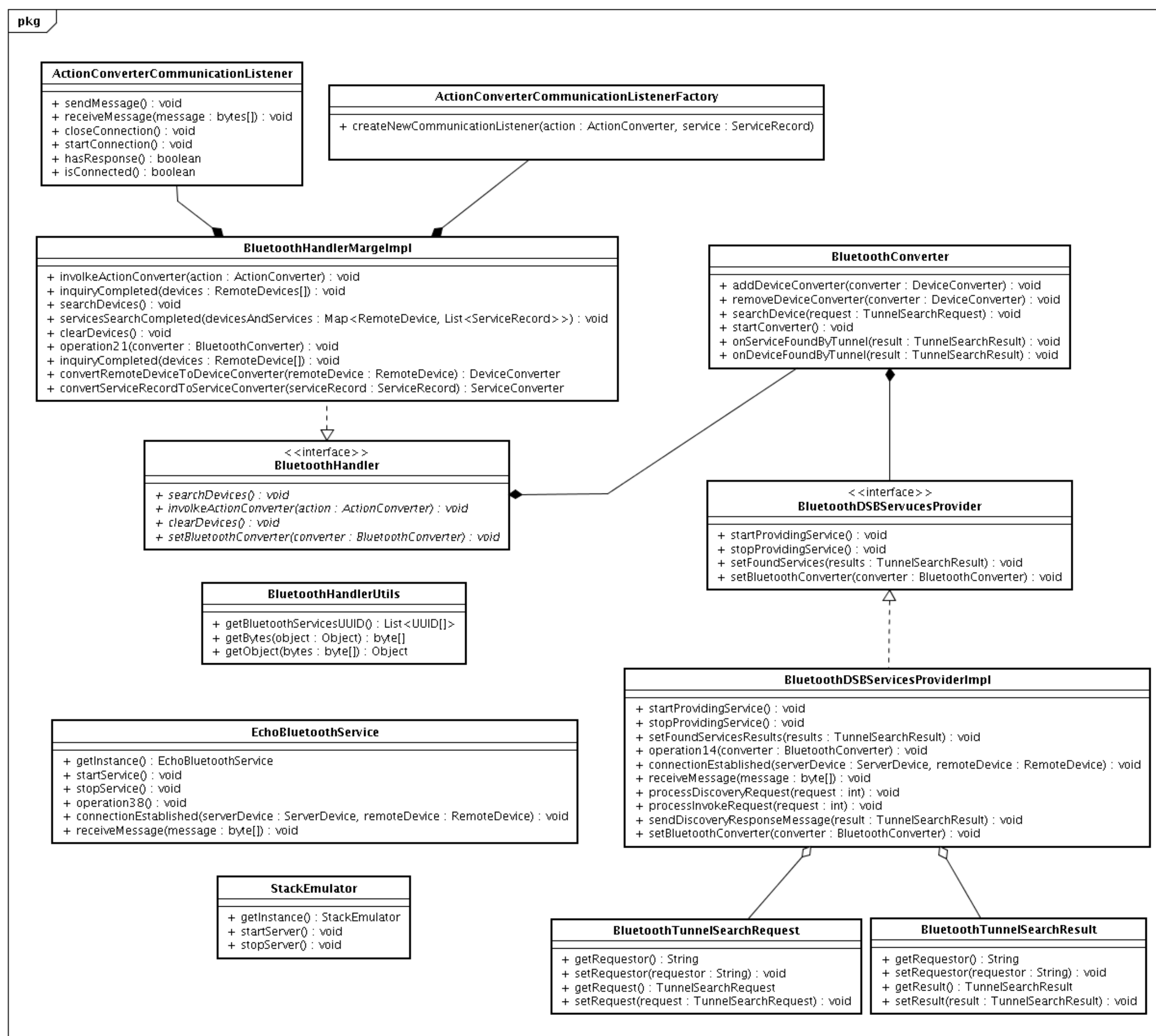



**Figura 6.1: Diagrama de classes do *BluetoothConverter***

## 6.2 Converter

A classe *Converter* (*br.ufsc.inf.ppgcc.converter.bluetooth.Converter*) define comportamentos que os *Converters* específicos das tecnologias devem implementar, e provê comunicação com as camadas superiores do DSB.

O *Converter* provê os seguintes métodos de comunicação com o DSB:

- *addDeviceConverter:* Adiciona um dispositivo virtual ao DSB;
- *removeDeviceConverter:* Remove um dispositivo virtual do DSB;
- *searchDevice*: Permite fazer uma busca de dispositivos ou serviços dentre todos os disponíveis em um DSB;

É necessário ainda que os seguinte métodos sejam implementados pelo *Converter* específico da tecnologia:

- *startConverter*: O DSB informa que deseja iniciar um *Converter* invocando este método. Aqui, funções de inicialização podem ser executadas, como uma busca inicial de serviços;
- *involkeActionConverter*: Este é o método que será invocado pelo DSB quando um serviço for chamado por um cliente. O *BluetoothConverter* interpretará esta chamada e fará uma invocação do serviço *Bluetooth* correspondente;
- *onServiceFoundByTunnel*: Este é um método de *callback*, que é invocado quando o DSB encontra algum serviço após uma busca iniciada por uma chamada ao método *searchDevice;*
- *onDeviceFoundByTunnel*: Este é um método de *callback*, que é invocado quando o DSB encontra algum dispositivo após uma busca iniciada por uma chamada ao método *searchDevice;*

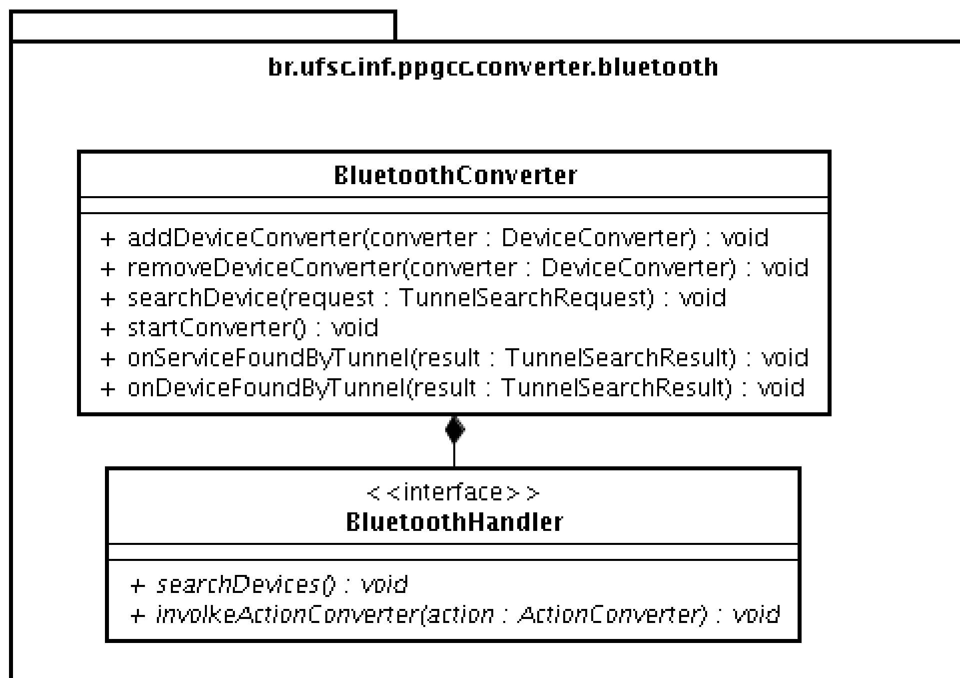



**Figura 6.2: Classes do pacote br.ufsc.inf.ppgcc.converter.bluetooth**

A figura 6.2 mostra o conteúdo do pacote *br.ufsc.inf.ppgcc.converter.bluetooth*, onde se encontra o *BluetoothConverter*. O *BluetoothConverter* comunica-se com dispositivos Bluetooth através da interface[3] *BluetoothHandler*. Essa interface especifica funções que serão de uso comum pelo *BluetoothConverter*, como busca e invocação de serviços Bluetooth. Desta forma, o *BluetoothConverter* não precisa se preocupar com a forma como *BluetoothHandler* estará implementado, desde que ele atenda as especificações da interface.

Como descrito na seção 4.2 - Java APIs for Bluetooth, a plataforma Java oferece comunicação com a tecnologia Bluetooth através da especificação JSR 82. Apesar de ser um padrão de desenvolvimento, a especificação trata a comunicação com dispositivos Bluetooth em sua forma natural. Sendo assim, o desenvolvedor deve conhecer bem os conceitos de desenvolvimento Bluetooth envolvidos, como a busca de serviços, parâmetros de comunicação e controle de conexão. Para facilitar o desenvolvimento do *BluetoothConverter*, utilizamos um *framework* que abstrai os conceitos de mais baixo nível envolvendo a tecnologia, e que permite concentrarmo-nos no desenvolvimento da aplicação em si – o *framework* Marge. Ainda, como implementação de testes de uma pilha Bluetooth foi utilizada a biblioteca Bluecove,

---

**3** Uma interface é um grupo de métodos sem corpo que definem o comportamento esperado para uma classe (SUN, 2009).

por ser de código aberto, executar em diversas plataformas e possuir facilitadores para desenvolvimento de aplicações, como um emulador de uma pilha Bluetooth.

### 6.2.1 Marge

Marge é um *framework*[4] que ajuda desenvolvedores a criar mais facilmente aplicações Bluetooth com Java. A ideia principal é facilitar a utilização da especificação JSR-82. O *framework* abstrai os conceitos de conexões, protocolos, troca de mensagens e busca por dispositivos e serviços. (MARGE, 2008a).

Buscas de serviços são realizadas alterando-se parâmetros de um objeto de configuração, como o UUID de um serviço, e passando este a classes específicas do *framework,* que encarregam-se de realizar a busca e retornar os resultados. Da mesma forma, ao encontrar um serviço, pode-se chamar uma das fábricas de comunicação do *framework*, que se responsabilizará por criar um canal de comunicação entre as tecnologias. O retorno de respostas, seja na busca de serviços ou na troca de mensagens, é feito através da utilização de *listeners*, onde define-se que uma determinada classe será responsável por tratar as respostas. Esta classe então fica no aguardo de um retorno, e assim que ele ocorrer esta classe encarrega-se de dar continuidade ao processo.

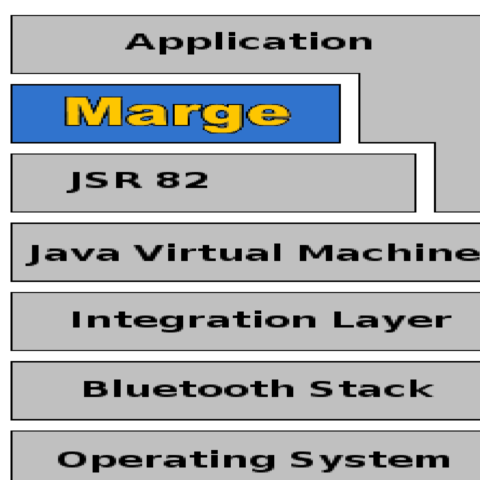



**Figura 6.3: Arquitetura do *framework* (MARGE, 2008b)**

4 Um *framework* é um conjunto de classes que constitui um projeto abstrato para a solução de uma família de problemas (FAYAD JOHNSON, 1999).

### 6.2.2 Bluecove

Bluecove é uma implementação de código aberto da especificação JSR-82, projetado para trabalhar com pilhas Bluetooth de plataformas Unix e Windows (BLUECOVE, 2009a). Ainda, a implementação fornece um emulador de uma pilha Bluetooth, permitindo desenvolver sem utilizar dispositivos reais. A implementação suporta os seguintes perfis Bluetooth:

- SDAP – *Service Discovery Application Profile*;
- RFCOMM – Protocolo de emulação de comunicação serial;
- L2CAP - *Logical Link Control and Adaptation Protocol*;
- OBEX – Perfil de troca de objetos genéricos;

A figura 6.4 apresenta a arquitetura da implementação Bluecove:

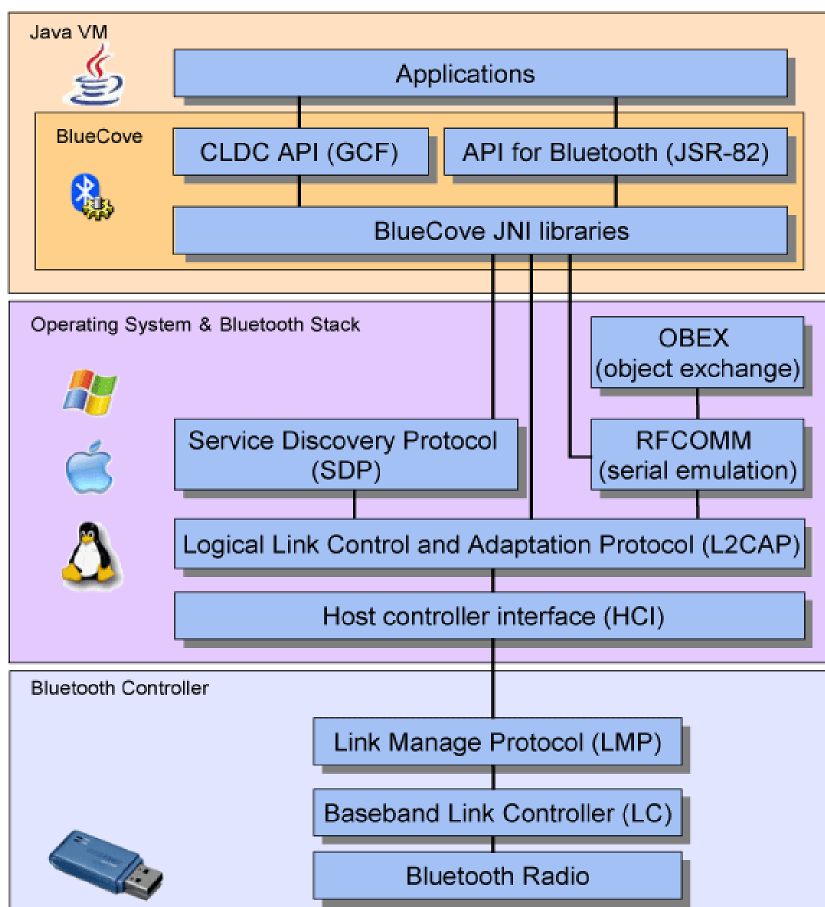



**Figura 6.4: Arquitetura Bluecove (BLUECOVE, 2009b)**

### 6.2.3 BluetoothHandlerMargeImpl

O gerenciador de comunicação Bluetooth foi implementado usando o *framework* Marge. A figura 6.5 apresenta as classes envolvidas no processo.

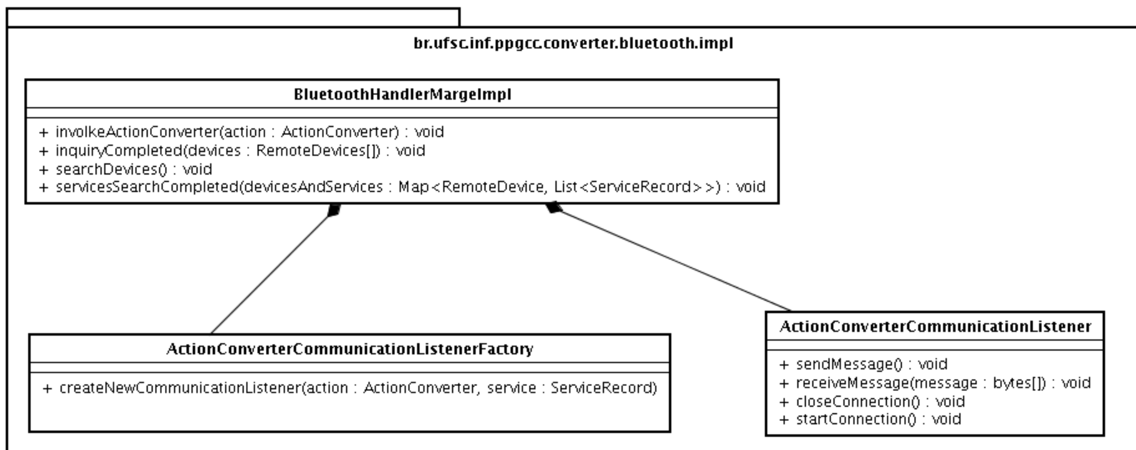



**Figura 6.5: Implementação da estrutura Bluetooth**

*BluetoothHandlerMargeImpl* é a implementação da interface *BluetoothHandler* utilizando o *framework* Marge como base. As classes *ActionConverterCommunicationListenerFactory* e *ActionConverterCommunicationListener* são classes auxiliares do *BluetoothHandlerMargeImpl*, que estendem as funções de comunicação do Marge. *ActionConverterCommunicationListenerFactory* é responsável por estabelecer comunicação entre um cliente e um servidor de um serviço Bluetooth, e *ActionConverterCommunicationListener* é o *listener* responsável por escutar a comunicação entre o cliente e o servidor.

A figura 6.6 apresenta a sequência de eventos durante uma busca de serviços Bluetooth pelo DSB:

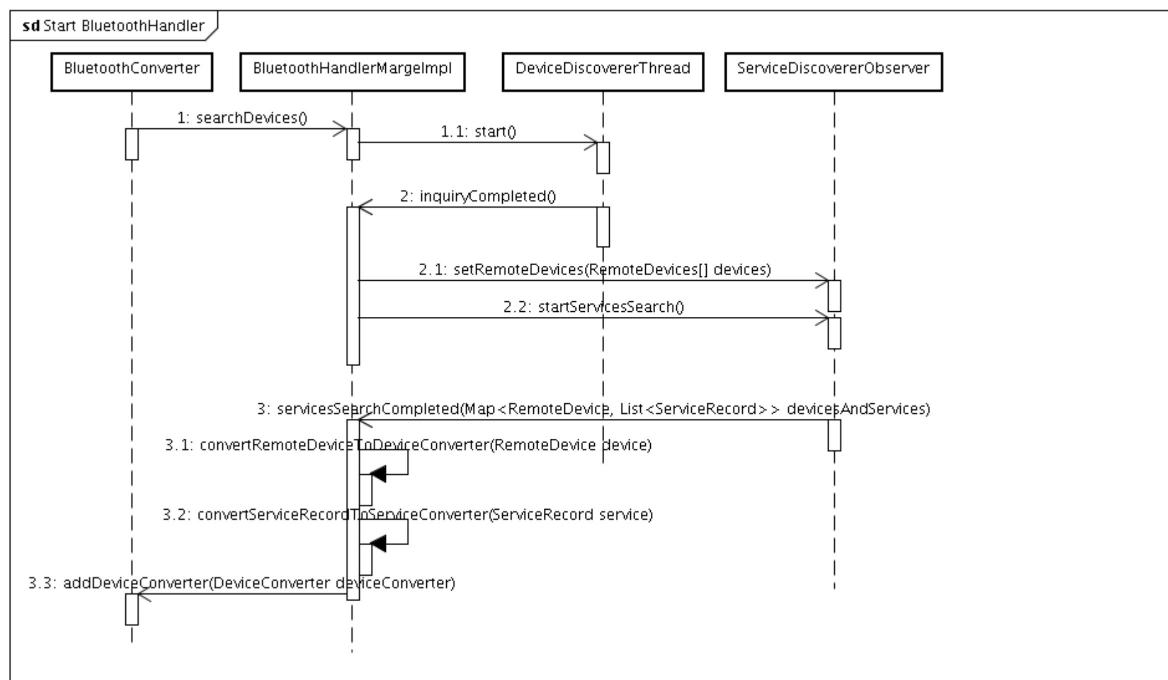



**Figura 6.6: Diagrama de sequência de uma busca de serviços**

O *BluetoothConverter* inicia a busca por serviços Bluetooth, passando a responsabilidade para o *BluetoothHandler*. O *BluetoothHandler* iniciar uma *thread* (*DeviceDiscovererThread*) de descobrimento, que irá buscar nos arredores dispositivos Bluetooth ativos. Ao terminar a busca, a *thread* devolve ao *BluetoothHandler* uma lista de dispositivos encontrados. O *BluetoothHandler* inicia então o processo de busca por serviços disponíveis, iniciando uma outra *thread* (*ServiceDiscovererObserver)*.

Esta *thread* cria outras *Threads*, uma para cada combinação de busca (para cada dispositivo, ela busca os serviços definidos no arquivo com os serviços a serem pesquisados, *bluetooth.properties*), e as observa durante a busca de serviços. Ao finalizar a busca de serviços, a *thread ServiceDiscovererObserver* repassa para o *BluetoothHandler* uma nova lista contendo todos os dispositivos e seus respectivos serviços encontrados.

Por fim, o *BluetoothHandler* deve converter os dispositivos e serviços encontrados para o formato esperado pelo DSB, e adicioná-los ao *Device Cache* do DSB. Durante o processamento das buscas, os resultados estão contidos em estruturas de dados definidas pelo pacote *javax.bluetooth*. As informações devem então ser extraídas e convertidas para as entidades DSB: *DeviceConverter, ServiceConverter, ActionConverter* e *ParameterConverter*.

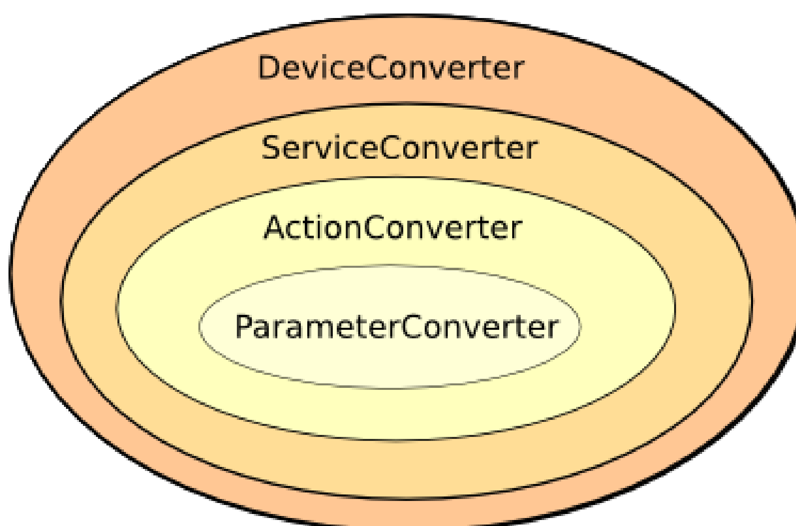



**Figura 6.7: Converters**

A figura 6.7 demonstra as camadas de *Converters* existentes. As entidades do DSB foram estruturadas de forma a abranger variadas estruturas de dispositivos e serviços. Cada entidade representa uma camada na estrutura de serviços: o *DeviceConverter* representa os dispositivos envolvidos (como um celular, uma impressora, etc); os *ServiceConverters* representam os serviços disponibilizados por um dispositivo (como fazer uma ligação, imprimir); os *ActionConverter* representam as ações que um serviço pode realizar (como imprimir colorido ou preto-e-branco); e os *ParameterConverters* representam os parâmetros que uma ação pode receber.

Para o *BluetoothConverter*, a conversão é feita da seguinte forma:

- Dispositivos Bluetooth são convertidos em *DeviceConverters*;
- Serviços Bluetooth são convertidos em *ServiceConverters* e *ActionConverters*, já que a especificação de *Actions* não se enquadra na arquitetura Bluetooth (não existe um nível acima de um serviço Bluetooth);
- Para a comunicação Bluetooth, foram definidos dois *ParameterConverters*, um parâmetro de entrada de dados (*BluetoothInputParam*), onde dados de entrada são envia-

dos em uma troca de mensagens Bluetooth, e um parâmetro de saída (*BluetoothOutputParam*), onde a resposta é recebida.

### 6.2.4 Invocação de um Serviço Bluetooth

A figura 6.8 apresenta a sequência de eventos em uma invocação de um serviço Bluetooth:

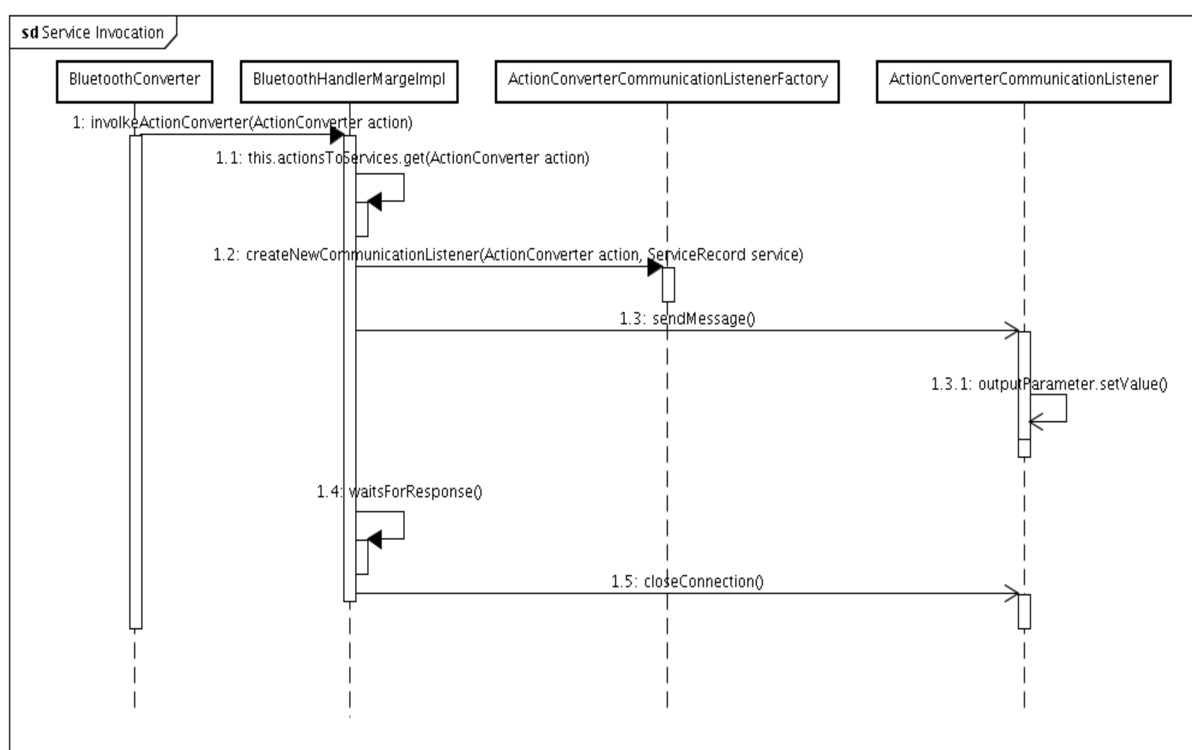



**Figura 6.8: Invocação de serviço**

O DSB recebe uma solicitação de invocação de um *ActionConverter*. Este, verificando que se trata de uma *Action* de um *BluetoothConverter*, repassa a invocação chamando o método *involkeActionConverter* do *BluetoothConverter*. O *BluetoothConverter* armazena um mapa (*Map*) que referencia *ActionConverters* a *ServiceRecords* – a entidade Bluetooth que representa um serviço. Com estas informações, o *Converter* cria uma entidade de comunicação *ActionConverterCommunicationListener* através da fábrica de comunicações *ActionConverterCommunicationListenerFactory*. Com a comunicação criada entre os dois pontos, um envio de mensagem acontece (*sendMessage)*. Como não é garantido que uma resposta ocorra,

devido à natureza *ad-hoc* do Bluetooth, um método assíncrono de espera foi implementado. O *BluetoothConverter* aguarda que o *ActionConverterCommunicationListener* registre uma resposta no parâmetro de saída até um tempo limite (*timeout*). Caso uma resposta não ocorra até o tempo limite, o parâmetro de saída recebe uma mensagem de erro. Por fim, o *ActionConverter* é retornado com o resultado de saída ao DSB, que o devolve a quem solicitou a invocação.

### 6.2.5 Descoberta e invocação de serviços por clientes Bluetooth

Para que dispositivos Bluetooth possam encontrar serviços disponibilizados pelo DSB, foi implementado um serviço Bluetooth a ser provido pelo DSB que possui esta característica. Este serviço está definido pela interface *br.ufsc.inf.ppgcc.converter.bluetooth.services.-BluetoothDSBServicesProvider* e é implementado em *br.ufsc.inf.ppgcc.converter.bluetooth.-services.impl.BluetoothDSBServicesProviderImpl*.

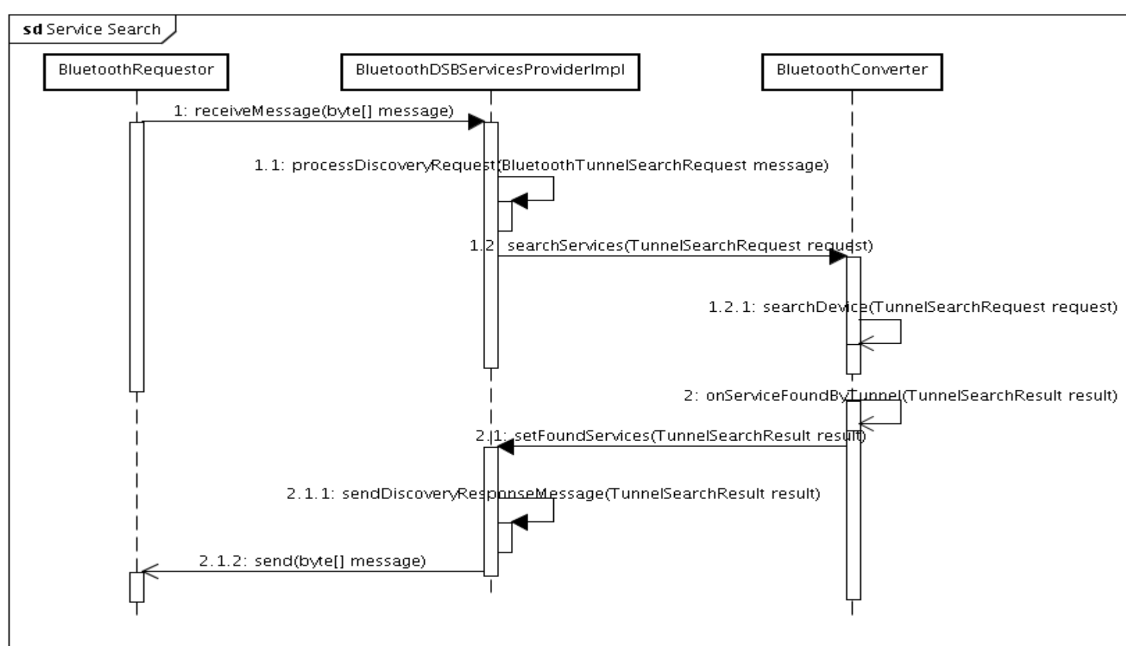



**Figura 6.9: Busca de serviços**

Neste contexto, um cliente Bluetooth se conecta ao serviço de busca, e envia uma mensagem ao serviço. Esta mensagem é um objeto de texto puro no formato XML como mostra a figura 6.10:

```xml
<TunnelSearchRequest>
    <requestorUUID>123456789</requestorUUID>
    <serviceToInvoke>EchoService</serviceToInvoke>
    <actionToInvoke>EchoService</actionToInvoke>
</TunnelSearchRequest>
```

**Figura 6.10: XML de busca de serviços**

Conforme mostrado na figura 6.9, ao receber a requisição, o serviço Bluetooth encaminha a solicitação ao DSB (*searchDevice*). O processo é assíncrono, então o serviço deve aguardar que o DSB finalize a solicitação. O DSB atende a solicitação invocando *onServiceFoundByTunnel,* devolvendo um objeto *br.ufsc.inf.ppgcc.search.TunnelSearchResult*, que encapsula a resposta de busca do DSB. Este objeto é devolvido ao cliente Bluetooth solicitante também com uma resposta em XML, e a partir deste resultado o serviço pode ser invocado pelo cliente Bluetooth, como mostra a figura 6.11:

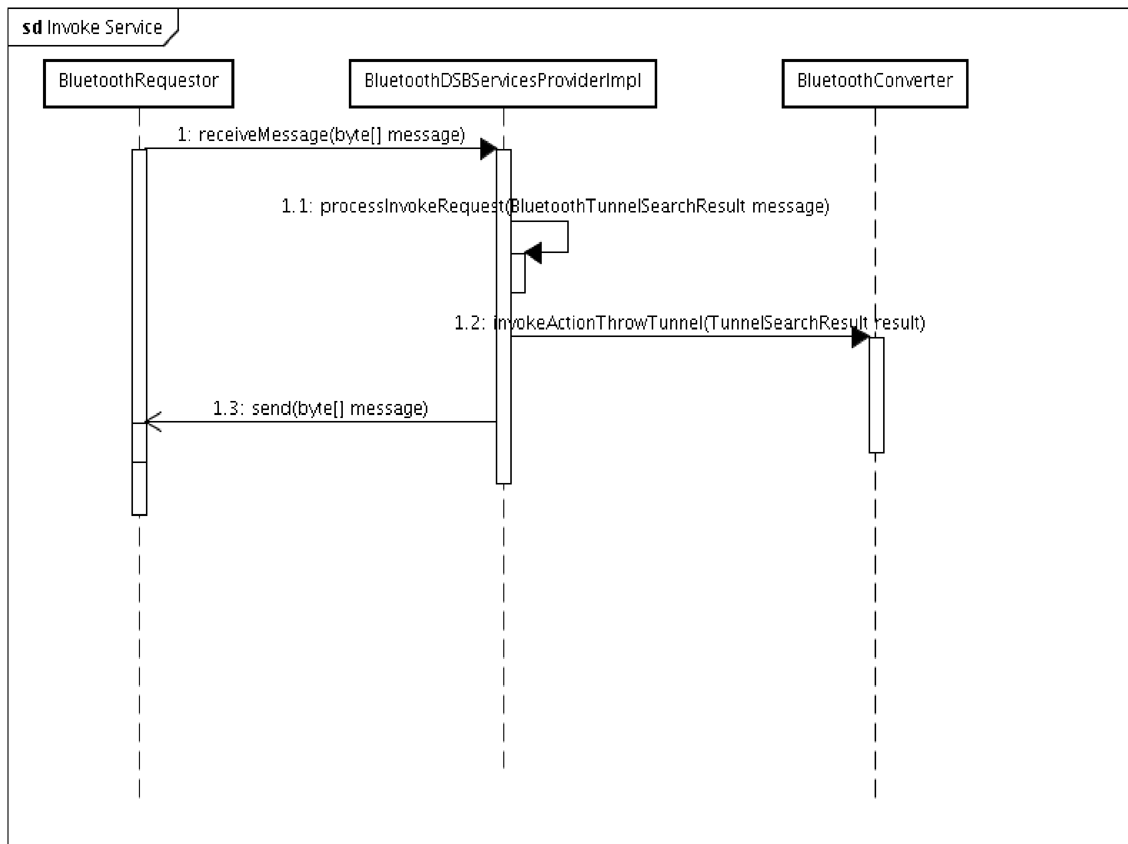



**Figura 6.11: Invocação de serviço**

O cliente ajusta os parâmetros do serviço, e realiza uma solicitação de invocação ao *BluetoothDSBServicesProviderImpl* passando um XML como o exemplo mostrado na figura 6.12:

```
<TunnelInvokeRequest>
    <requestorUUID>12345</requestorUUID>
    <serviceToInvoke>Namespace</serviceToInvoke>
    <actionToInvoke>Porttype</actionToInvoke>
    <parameters>
       <parameter name="paramName" value="paramValue"/>
    </parameters>
</TunnelInvokeRequest>
```

**Figura 6.12: XML de invocação de serviço**

O DSB processará a requisição e devolverá um *ActionConverter* com o resultado da operação. O resultado é então devolvido ao cliente Bluetooth na forma de um objeto XML com os dados de resposta.

## 6.3 Testes

O ambiente demonstrado na figura 6.13 foi utilizado para testes do *BluetoothConverter*.

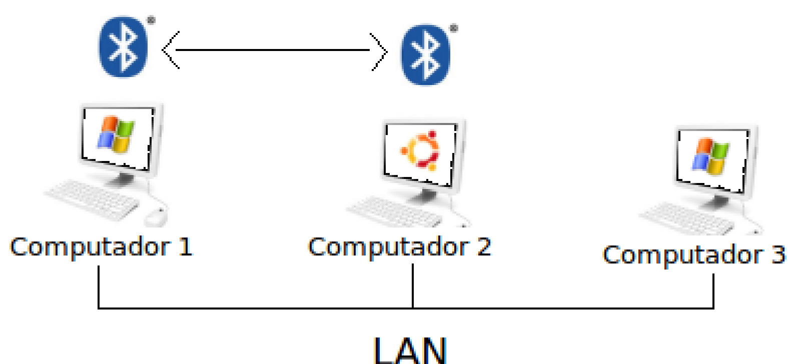



**Figura 6.13: Arquitetura de testes**

- Computador 1: Athlon XP 2Ghz, 256mb RAM, Hd de 80Gb, sistema operacional Windows XP com SP3, com adaptador Bluetooth instalado com o driver fornecido pelo próprio Windows.

- Computador 2: Intel Core2Duo 1.7Ghz, 2Gb RAM, HD 160Gb, sistema operacional Ubuntu 9.04, com adaptador Bluetooth instalado com a pilha Bluetooth para Linux Bluez.

- Computador 3: Intel Core2Quad 2.4Ghz, 2Gb RAM, HD 250Gb, sistema operacional Windows XP.

### 6.3.1 Teste 1: DSB acessando um serviço Bluetooth

Este teste consiste em utilizar o *BluetoothConverter* para invocar um serviço Bluetooth sendo provido por um DSB. Para isto, a seguinte arquitetura de software foi utilizada:

- Computador 1: Contém um serviço Bluetooth ativado e aguardando utilização. Um serviço simples foi implementado, onde sua única tarefa é retornar ao requisitante a mesma mensagem recebida pelo serviço;

- Computador 2: Contém o DSB com o *BluetoothConverter* em execução;

- Computador 3: Contém o DPWS Explorer (WS4D, 2009), um aplicativo que permite visualizar e executar chamadas a serviços disponibilizados em um ambiente WS4D.

Neste contexto, o *BluetoothConverter* encontra o serviço Bluetooth e disponibiliza ao DSB. Assim, o DPWS Explorer consegue visualizar o serviço. O teste consiste em enviar uma chamada de invocação ao serviço e calcular o tempo de resposta. As mensagens são enviadas com tamanho fixo, iniciando com 10kb e aumentando em 10kb a cada nova chamada, até chegar a 100kb. Os seguintes resultados foram obtidos:

| Tamanho da mensagem | Tempo de resposta em segundos (médio) |
|---|---|
| 10kb | 2,063667 |
| 20kb | 3,105667 |
| 30kb | 4,249000 |
| 40kb | 5,333667 |
| 50kb | 5,837000 |
| 60kb | 5,363333 |
| 70kb | 6,504333 |
| 80kb | 7,804333 |
| 90kb | 11,086000 |
| 100kb | 12,488000 |

**Tabela 1: Resultados do teste 1**

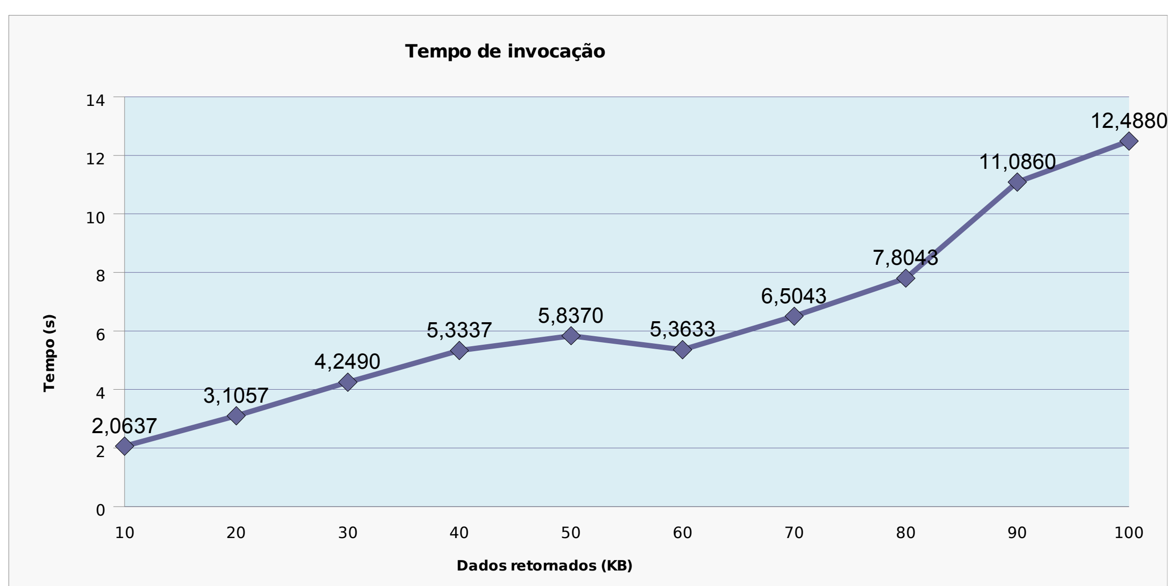



**Figura 6.14: Gráfico do tempo de invocação do teste 1**

### 6.3.2 Teste 2: Cliente Bluetooth acessando um serviço do DSB

Este teste consiste em ter um serviço sendo provido pelo DSB, independente de tecnologia, que ficará disponibilizado para acesso por clientes Bluetooth através do serviço do *BluetoothConverter BluetoothDSBServicesProvider*. Para isto, a seguinte arquitetura de software foi utilizada:

- Computador 1: Contém o DSB com o *BluetoothConverter* e o *BluetoothDSBServicesProvider* em execução, provendo um serviço para fins de teste que simplesmente retorna o valor recebido;
- Computador 2: Contém um cliente Bluetooth que irá acessar o serviço *BluetoothDSBServicesProvider.*

Neste contexto, o *BluetoothDSBServicesProvider* provê um serviço Bluetooth que permite que clientes Bluetooth encontrem e invoquem serviços do DSB. O teste consiste em enviar uma chamada de invocação ao serviço e calcular o tempo de resposta. As mensagens são enviadas com tamanho fixo, iniciando com 10kb e aumentando em 10kb a cada nova chamada, até chegar a 100kb. Os seguintes resultados foram obtidos:

| Tamanho da mensagem | Tempo de resposta em segundos (médio) |
|---|---|
| 10kb | 2,0109 |
| 20kb | 3,9580 |
| 30kb | 5,0013 |
| 40kb | 5,9801 |
| 50kb | 6,3368 |
| 60kb | 6,3490 |
| 70kb | 7,9990 |
| 80kb | 8,3444 |
| 90kb | 10,2978 |
| 100kb | 11,6671 |

**Tabela 2: Resultados do teste 2**

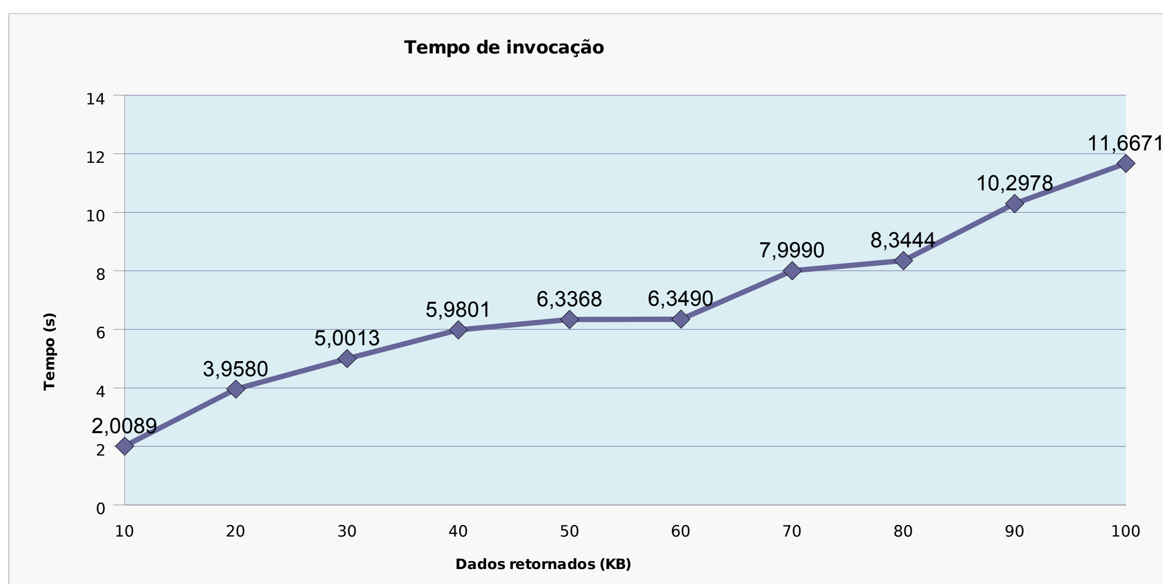



**Figura 6.15: Gráfico do tempo de invocação do teste 2**

### 6.3.3 Avaliação

Os testes demonstraram a utilização do *BluetoothConverter* em um ambiente real, e os resultados obtidos foram satisfatórios e dentro do esperado. O *BluetoothConverter* age sem muita interferência no processo de comunicação. As demoras no tempo de resposta são da própria natureza da comunicação com a tecnologia Bluetooth, onde conforme a quantidade de dados é maior, também cresce o tempo de transmissão de uma ponta a outra.

# 7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O momento crescente de dispositivos em ambientes ubíquos exige estudo de soluções para integração de tecnologias. O DPWS surge como opção de integração para este este tipo de ambiente, e o DSB surge como uma camada alternativa que permite integrar dispositivos de recursos restritos ao ambiente DPWS. Ambos seguem o processo contínuo de maturação, mas apresentam-se como opções viáveis.

O *BluetoothConverter* segue o mesmo caminho de maturação. Para o propósito deste trabalho, acreditamos que o objetivo principal foi atingido, que é o desenvolvimento de uma camada que permita integrar dispositivos Bluetooth a um ambiente de web services. O *BluetoothConverter* foi desenvolvido e mostra-se funcional. Foi possível integrar um serviço Bluetooth simples, de forma que este possa ser invocado por outros dispositivos não Bluetooth, assim como um dispositivo Bluetooth pôde se comunicar com o DSB para invocar outros serviços. Os testes apresentaram resultados satisfatórios, onde o DSB e o *BluetoothConverter* em si não prejudicaram o tempo de resposta, ocorrendo apenas a demora no tempo de resposta por causa da velocidade de transmissão que é natural da tecnologia Bluetooth.

Algumas limitações encontradas durante o desenvolvimento deste trabalho podem ainda servir de inspiração para o desenvolvimento de trabalhos futuros.

- O *BluetoothConverter* neste momento é limitado a tratar apenas uma conexão Bluetooth por vez, isto porque existe uma limitação da tecnologia onde apenas uma conexão pode existir em um determinado instante. Isto implica que não é possível, por exemplo, atender a dois dispositivos Bluetooth simultaneamente, ou atender a uma requisição de um dispositivo Bluetooth e um chamado DSB para outro dispositivo Bluetooth. Desta forma, o estudo de uma alternativa para o tratamento de conexões diferentes é um trabalho a ser avaliado;

- O *BluetoothConverter* foi testado com a execução de serviços Bluetooth simples, como serviços de eco que retornavam a mesma mensagem enviada. Uma possibilidade é estudar a integração do *BluetoothConverter* com um dos perfis de serviços Bluetooth, como a utilização de um serviço de impressão;

- O *BluetoothConverter* foi desenvolvido utilizando as tecnologias Java, porém com o avanço do desenvolvimento do DSB para outras tecnologias, a conversão do *BluetoothConverter* para estas outras tecnologias também podem ser uma fonte de estudo.

# 8 REFERÊNCIAS BIBLIOGRÁFICAS

APPLE2, 2008b: Apple, Bluetooth Architecture, 2008, http://developer.apple.com/documentation/DeviceDrivers/Conceptual/Bluetooth/art/bt_protocol_stack.gif

ARAUJO SIQUEIRA, 2008: Gustavo Medeiros Araújo, Frank Siqueira, The Device Service Bus: A solution for Embedded Device Integration throught Web Services, 24rd ACM Symposium on Applied Computing, 2008

AYALA BROWNE et al, 2002: Dietrich Ayala, Christopher Browne, Vivek Choppra, Poornachandra Sarang, Kapil Apshankar, Tim Mcallister, Professional Open Source Web Services, 2002

BLUECOVE, 2009a: Bluecove, Bluecove – JSR-82 project, 2009, http://www.bluecove.org

BLUECOVE, 2009b: SUN, The Java ME Platform, 2008, http://java.sun.com/javame/index.jsp

BLUETOOTH SIG, 2008a: Bluetooth SIG, How Bluetooth Technology works, 2008, http://www.bluetooth.com/Bluetooth/Technology/Work

BLUETOOTH SIG, 2008b: Bluetooth SIG, Piconet image, 2008, http://www.bluetooth.com/NR/rdonlyres/4124417B-EA2F-4EA8-B4BE-4EC08A05BFD3/6974/Piconets.jpg

BLUETOOTH SIG, 2009: Bluetooth SIG, Profiles Overwiew, 2008, http://www.bluetooth.com/Bluetooth/Technology/Works/Profiles_Overview.htm

EETIMES, 2008: EETimes, How Bluetooth got it's name, 2008, http://www.eetimes.eu/scandinavia/206902019?cid=RSSfeed_eetimesEU_scandinavia

FAYAD JOHNSON, 1999: Mohamed E. Fayad, Ralph E. Johnson, Domain-Specific Application Frameworks, 1999

IBM, 2009: IBM, Defining SOA as an Architectural Style, 2009, http://www.ibm.com/developerworks/library/ar-soastyle/

INTERNATIONAL TELECOMMUNICATION, 2008: International Telecommunication Union, Frequently asked question, 2008, http://www.itu.int/ITU-R/terrestrial/faq/index.html

JAVA COMMUNITY PROCESS, 2008: Java Community Process, The Java Community Process, 2008, http://jcp.org/en/introduction/faq

JAVANET, 2008: Java.net, Duke, 2008, https://duke.dev.java.net/images/glassfish/Duke.wave.shadow.gif

MARGE, 2008a: Marge Dev, Home, 2008, https://marge.dev.java.net

MARGE, 2008b: Marge Dev, Arquitecture, 2008, https://marge.dev.java.net/img/marge-arch.png

MSDN, 2008a: Microsoft, A Technical Introduction to the Devices Profile for Web Services, 2008, http://msdn.microsoft.com/en-us/library/ms996400.aspx

MSDN, 2008b: Microsoft, Communication Example, 2008, http://i.msdn.microsoft.com/ms996400.deviceprofile-techoverview01(en-us,MSDN.10).gif

OPEN GROUP, 2008a: Open Group, Service Oriented Architecture, 2008, http://opengroup.org/projects/soa/doc.tpl?gdid=10632

OPEN GROUP, 2009: Open Group, SOA Source Group, 2009, http://www.opengroup.org/projects/soa-book/

PALO, 2008: Palo Alto Research Center, Ubiquotous Computing, 2008, http://www.ubiq.com/hypertext/weiser/UbiHome.html

SINGH BRYDON et al, 2004: Inderjeet Singh, Sean Brydon, Greg Murray, Vijay Ramachandran, Thierry Violleau, Beth Stearns, Designing Web Services with the J2EE 1.4 Plataform, 2004

SUN DEVELOPERS, 2008: SUN Developers, The Java API for Bluetooth Tecnology, 2008, http://developers.sun.com/mobility/midp/articles/bluetooth2

SUN, 2008a: SUN, Java Technology: The Early Days, 2008, http://java.sun.com/features/1998/05/birthday.html

SUN, 2008b: SUN, *7 display, 2008, http://java.sun.com/features/1998/05/images/star7.3.small.jpg

SUN, 2008c: SUN, Java SE Overview, 2008, http://java.sun.com/javase/6/features.jsp

SUN, 2008d: SUN, Java EE at Glance, 2008, http://java.sun.com/javaee

SUN, 2008e: SUN, Company Info, 2008, http://www.sun.com/aboutsun/company/index.jsp

SUN, 2009: SUN, What is an Interface?, 2009, http://java.sun.com/docs/books/tutorial/java/concepts/interface.html

W3 SCHOOLS, 2008a: W3Schools, Introduction to XML Schema, 2008, http://www.w3s-chools.com/Schema/schema_intro.asp

W3, 2009: World Wide Web Consortium, Web Service Definition Language, 2009, http://www.w3.org/TR/wsdl

W3C, 2008a: World Wide Web Consortium, Web Services Activity, 2008, http://www.w3.org/2002/ws

W3C, 2009: World Wide Web Consortium, SOAP Version 1.2 Part I, 2009, http://www.w3.org/TR/soap12-part1/

W3C, 2009b: World Wide Web Consortium, Service Oriented Architecture, 2009, http://www.w3.org/2003/Talks/0521-hh-wsa/slide5-0.html

WS4D, 2009: Web Services for Devices, DPWS Explorer, 2009, http://ws4d.e-technik.uni-rostock.de/?cat=4

ZEEB BOBEK et al, 2007: Elmar Zeeb, Andreas Bobek, Frank Golatowski, Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services, , 2007

Zigbee, 2009: Zigbee, Zigbee Alliance, 2009, http://www.zigbee.org

# ANEXO A – ARTIGO

## Integração de Dispositivos Móveis com Web Services através de Bluetooth

**Fabio Kreusch**[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
UFSC-CTC-INE
CEP 88040-900 - Campus Universitário Cx.P. 476
Florianópolis – SC - Brasil

fabiokr@inf.ufsc.br

***Abstract.*** *This work aims to study ways to integrate systems in the ubiquitous context of the Bluetooth technology. For that, integration technologies like web services are analyzed and an integration layer between the Bluetooth technology and a web services architecture are developed. The developing of this integration layer aims to make possible that limited devices be part of an integrated ubiquitous environment, independently of the used technologies.*

***Resumo.*** *Este trabalho de conclusão de curso visa estudar formas de interação entre sistemas no contexto ubíquo da tecnologia Bluetooth. Para isto, tecnologias de integração como serviços web são analisadas e uma camada de integração entre a tecnologia Bluetooth e uma arquitetura de serviços web é desenvolvida. O desenvolvimento desta camada de integração visa permitir que dispositivos de*

*recursos limitados possam fazer parte de um ambiente ubíquo integrado, independente das tecnologias de comunicação envolvidas.*

## 1. Introdução

A evolução da computação em todas as suas faces tem criado um novo ambiente propício à comunicação constante entre diversos dispositivos, independentemente de sua localização. Telefones celulares comunicam-se com outros através de ondas eletromagnéticas, computadores comunicam- se entre si e com outros dispositivos através da internet, pequenos dispositivos comunicam-se através de pequenas redes ad-hoc. Estes exemplos compartilham um objetivo fim, o de realizar alguma tarefa para um ser humano, seja executar a comunicação entre pessoas, ou prover alguma funcionalidade que seja útil ao usuário.

Este conceito de comunicação ocorrendo em nosso meio, muitas vezes sem a percepção imediata de que o processo envolve alguma computação, é conhecido como computação ubíqua (*Ubiquitous Computing*), termo cunhado pelo pesquisador Mark Weiser [PALO, 2008], e que vem recebendo atenção cada vez maior por parte da comunidade acadêmica e da indústria.

Aliado a conceitos de intercomunicação, temos também acompanhado o crescimento do estudo e utilização da arquitetura de serviços web. Serviços web (do inglês, *web* services) visam facilitar a troca de informação entre diferentes aplicações [W3C, 2008a], independente da plataforma onde a aplicação é executada (dispositivos móveis ou não, sistemas operacionais, etc) e independente da linguagem em que o aplicativo foi desenvolvido. Para isto, padrões de troca de informação foram definidos, como a utilização do protocolo SOAP e documentos XML.

Este trabalho se propôs a estudar e especificar mecanismos para comunicação entre dispositivos móveis e baseados em serviços web utilizando o Bluetooth [BLUETOOTH SIG, 2008a], uma tecnologia que tem sido largamente adotada em diversos dispositivos móveis, por suas características de robustez, baixo consumo de energia e baixo custo. Para este propósito, será estendida a arquitetura DSB [ARAUJO SIQUEIRA, 2008], que prevê um meio de intercomunicação entre dispositivos de várias tecnologias. Neste trabalho, uma extensão que permita integrar um dispositivo com suporte a Bluetooth será desenvolvida e adaptada para funcionar junto ao DSB.

# 2. Web Services

## 2.1. SOA

SOA (*Service-Oriented Architecture)* é um estilo de arquitetura genérico, um paradigma conceitual, que suporta orientação a serviços. Orientação a serviços é uma maneira de desenvolver software, onde existem serviços que podem ser utilizados por clientes e estes retornam uma resposta. De forma geral, um serviço é uma atividade repetível que produz um resultado específico (como por exemplo, validar um número de cartão de crédito) [OPEN GROUP, 2009].

Do ponto de vista de negócios, a arquitetura SOA pode ser definida como "um conjunto de serviços que pode ser utilizado na construção de soluções e que pode ser exposto a clientes e parceiros". Do ponto de vista de arquitetura de software, pode ser definido como um "conjunto de princípios e padrões para prover as seguintes características a um software: modularidade, encapsulamento, baixo acoplamento, reutilização e composição". [IBM, 2009]

Um *web service*(serviço web) é um software (uma aplicação) acessível através de uma rede (internet, intranet, etc.) e identificado por meio de uma URI (*Universal Resource Identifier*[5]), que é acessada por clientes utilizando protocolos baseados em XML. [SINGH BRYDON et al, 2004].

Os seguintes benefícios provocaram um aumento de interesse pela utilização de *web services*:

Interoperabilidade: desenvolvedores de aplicações costumam criar seus sistemas em diferentes ambientes, linguagens e arquiteturas. Muitas vezes surge a necessidade de intercomunicação entre estas aplicações. Os *web services* oferecem interoperabilidade entre estes sistemas, independente da forma como foram desenvolvidos.

---

5 Uma URI é um endereço que identifica um recurso em uma rede, sendo imagens, textos, serviços, etc., que tornam um recurso disponível através de um método de acesso, como HTTP ou FTP (W3C, 2008).

Serviços de negócios através da Web: os *web services* permitem às empresas disponibilizar seus serviços através da web para qualquer cliente interessado, como, por exemplo, uma listagem de produtos.

Integração de sistema existentes: muitas vezes, empresas possuem aplicações legadas onde o custo de remanejamento para uma nova tecnologia pode ser muito alto. *Web services* permitem integrar sistemas legados e disponibilizar seus serviços já existentes para outros interessados sem a necessidade de remanejamento da aplicação.

Liberdade de escolha: *web service* é um padrão que não é ligado a uma empresa fornecedora específica. Várias empresas desenvolvem soluções e ferramentas para facilitar o desenvolvimento de *web services*, permitindo aos clientes escolherem aquela que melhor se adequa a sua necessidade.

Suporte a um maior número de clientes: *web services* podem ser acessados por variados tipos de clientes, desenvolvidos em diferentes linguagens.

## 3. Bluetooth

Bluetooth é uma tecnologia de comunicação sem fio a curta distância, que visa substituir a utilização de cabos para a comunicação entre dispositivos eletrônicos através da aplicação de uma arquitetura WPAN. Outro exemplo de tecnologia que utiliza a arquitetura WPAN é o Zigbee [Zigbee, 2009].

A tecnologia Bluetooth foi baseada nas seguintes premissas especificadas pelo Bluetooth SIG:

Suporte a dados e voz: a tecnologia deve ser capaz do prover transmissão de dados e voz com boa qualidade, sendo considerada de boa qualidade a atual transmissão de voz por telefones com fio.

Permitir conexões *ad-hoc*: a natureza dinâmica dos dispositivos móveis torna complexa a tarefa de assumir qualquer informação referente ao ambiente de operação. Assim sendo, a tecnologia deve poder detectar e estabelecer conexões com qualquer outra unidade compatível.

Poder lidar com interferência causada por outras fontes: o Bluetooth opera em ondas de rádio na frequência de 2.4GHz, que é utilizada por vários outros dispositivos, por ser uma frequência de rádio não licenciada. A tecnologia deverá saber lidar com possíveis interferências causadas por estes dispositivos.

Utilização mundial: a tecnologia deve se adequar às diferenciações relativas à variação de rádio frequência que ocorre em vários países.

Proteção similar à de uma transmissão de um cabo: a tecnologia deve prover mecanismos de segurança e autenticação, visto que seus usuários não desejarão que seus dados possam ser interceptados por outros receptores diferentes do escolhido pelo usuário.

Tamanho pequeno: o módulo de rádio deve ser pequeno o bastante para poder ser utilizado em vários dispositivos portáteis.

Baixo consumo de energia: vários dos dispositivos que utilizarão a tecnologia serão movidos a bateria. Esta característica implica que a tecnologia não deverá comprometer o tempo de vida das baterias dos dispositivos.

# 4. Java

Java refere-se basicamente a três frentes de tecnologia: a linguagem de programação Java, a máquina virtual Java e a plataforma de desenvolvimento Java. A linguagem de programação Java tem sua sintaxe baseada principalmente em C++. É uma linguagem de programação orientada a objetos, utilizando os conceitos de classes (definições das características de alguma coisa, como suas propriedades e seus comportamentos), objetos (um exemplar de uma classe), métodos (comportamentos), herança (versões especializadas de classes herdam as características de suas classes pais) e polimorfismo (um objeto pode ser tratado como de outra classe para que realize um comportamento diferente de seu comportamento original).

Para utilização da tecnologia Bluetooth juntamente com a plataforma Java, foi criada a especificação JSR 82: *Java APIs for Bluetooth*, com a finalidade de padronizar o desenvolvimento de aplicações Bluetooth para a plataforma. A especificação visa remover a complexidade de se lidar com a pilha de protocolos Bluetooth, e permitir que desenvolvedores foquem no desenvolvimento de aplicações. A especificação é mantida pelo

JCP (*Java Community Process*), um grupo com o objetivo de desenvolver e revisar especificações relativas à plataforma Java [JAVA COMMUNITY PROCESS, 2008].

# 5. Integração de dispositivos com Web Services

A computação ubíqua tem recebido um forte foco de estudo atualmente. Temos acompanhado a evolução dos sistemas computacionais de forma que o modelo de computação centralizada tem sido substituído por um modelo de computação distribuída. Neste contexto, dispositivos de diferentes categorias, desde computadores interligados por meio de redes tradicionais até dispositivos móveis como celulares, sensores móveis e tags RFID, podem fazer parte de um mesmo ambiente.

Estes novos dispositivos que hoje fazem parte de nosso dia-a-dia têm recebido novas características, como aumento de poder de processamento e funcionalidades de comunicação com redes *wireless*, permitindo a comunicação com outros dispositivos, ou até mesmo executar protocolos de comunicação como os requeridos pela arquitetura SOA. Apesar do aumento das capacidades destes dispositivos, a falta de padronização tem dificultado a intercomunicação entre eles. Em meio a isto, uma das propostas de intercomunicação entre dispositivos é o DPWS, um padrão baseado em serviços Web, proposto por um grupo de empresas, que visa permitir a interoperabilidade entre dispositivos, não limitando-se a determinadas características de rede ou de plataforma computacional.

O DPWS é um padrão de comunicação para dispositivos de recursos restritos, baseado no padrão de disponibilização de serviços proposto pelos serviços web. O DPWS organiza um conjunto de mecanismos empregados pelos serviços web de forma a adequá-los à utilização por dispositivos de recursos restritos, e é uma especificação para aqueles que estão desenvolvendo bibliotecas de comunicação de dispositivos em rede [MSDN, 2008a].

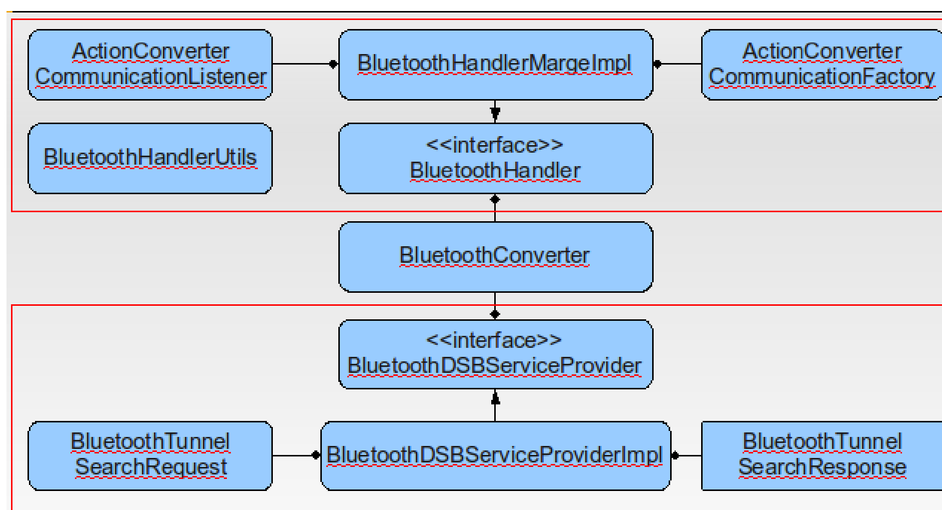
A arquitetura proposta pelo DPWS é robusta e trata vários aspectos referentes à utilização de serviços web por dispositivos. Porém, nem todo dispositivo tem a capacidade de executar uma implementação da arquitetura, principalmente em se tratando de dispositivos de recursos limitados, como celulares e leitores RFID. Em todo caso, muitas vezes estes dispositivos possuem alguma tecnologia de comunicação como, por exemplo, RFID e Bluetooth.

O *Device Service Bus* é uma camada de infra-estrutura baseada em uma plataforma leve e portável, que pode ser executada em estações de trabalho e pequenos dispositivos. Seguindo os princípios do SOA, o DSB provê um meio de integrar dispositivos, agindo como um intermediário entre provedores e consumidores de serviços. [ARAUJO SIQUEIRA, 2008].

## 6. Integração da tecnologia Bluetooth ao DSB

O objetivo final deste trabalho foi desenvolver um *Converter* que permita que dispositivos com a tecnologia Bluetooth possam fazer parte da arquitetura DSB. Para isto, foi desenvolvido o *BluetoothConverter*, um *Converter* seguindo as características do DSB que será responsável por interagir com a tecnologia Bluetooth.

**Figura 1: Diagrama de classes simplificado**




A classe *Converter* (*br.ufsc.inf.ppgcc.converter.bluetooth.Converter*) define comportamentos que os *Converters* específicos das tecnologias devem implementar, e provê comunicação com as camadas superiores do DSB. O *BluetoothConverter* extende um *Converter,* e comunica-se com dispositivos Bluetooth através da interface[6] *BluetoothHandler*. Essa interface especifica funções que serão de uso comum pelo *BluetoothConverter*, como

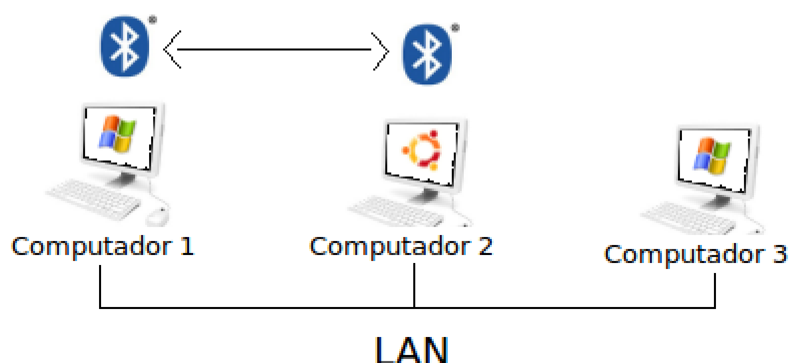
6 Uma interface é um grupo de métodos sem corpo que definem o comportamento esperado para uma classe [SUN, 2009].

busca e invocação de serviços Bluetooth. Desta forma, o *BluetoothConverter* não precisa se preocupar com a forma como *BluetoothHandler* estará implementado, desde que ele atenda as especificações da interface.

## 6.1 Testes

O ambiente demonstrado na figura abaixo foi utilizado para testes do *BluetoothConverter*.

**Figura 2: Ambiente de testes**




- Computador 1: Athlon XP 2Ghz, 256mb RAM, Hd de 80Gb, sistema operacional Windows XP com SP3, com adaptador Bluetooth instalado com o driver fornecido pelo próprio Windows.

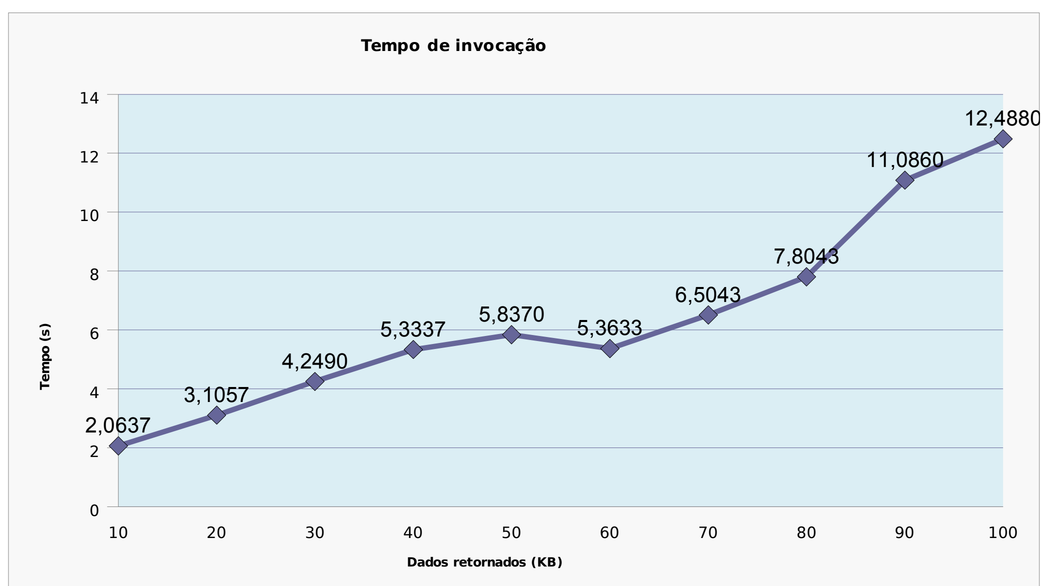
Computador 2: Intel Core2Duo 1.7Ghz, 2Gb RAM, HD 160Gb, sistema operacional Ubuntu 9.04, com adaptador Bluetooth instalado com a pilha Bluetooth para Linux Bluez.

- Computador 3: Intel Core2Quad 2.4Ghz, 2Gb RAM, HD 250Gb, sistema operacional Windows XP.

Teste 1: DSB acessando um serviço Bluetooth: Este teste consiste em utilizar o *BluetoothConverter* invocar um serviço Bluetooth sendo provido por um DSB. Neste contexto, o *BluetoothConverter*encontra o serviço Bluetooth e disponibiliza ao DSB. Assim,

o DPWS Explorer consegue visualizar o serviço. O teste consiste em enviar uma chamada de invocação ao serviço e calcular o tempo de resposta. As mensagens são enviadas com tamanho fixo, iniciando com 10kb e aumentando em 10kb a cada nova chamada, até chegar a 100kb. Os seguintes resultados foram obtidos:

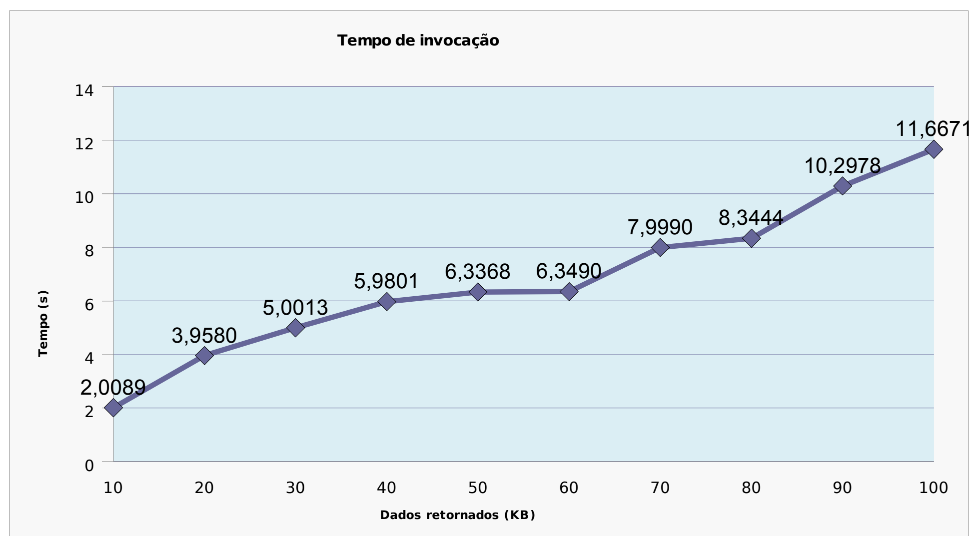
**Figura 3: Resultado do teste 1**




Teste 2: Cliente Bluetooth acessando um serviço do DSB. Este teste consiste em ter um serviço sendo provido pelo DSB, independente de tecnologia, que ficará disponibilizado para acesso por clientes Bluetooth através do serviço do *BluetoothConverter BluetoothDSBServicesProvider*.

Neste contexto, o *BluetoothDSBServicesProvider* provê um serviço Bluetooth que permite que clientes Bluetooth encontrem e invoquem serviços do DSB. O teste consiste em enviar uma chamada de invocação ao serviço e calcular o tempo de resposta. As mensagens são enviadas com tamanho fixo, iniciando com 10kb e aumentando em 10kb a cada nova chamada, até chegar a 100kb. Os seguintes resultados foram obtidos:

**Figura 4: Resultados do teste 2**




# 7. Conclusão

O momento crescente de dispositivos em ambientes ubíquos exige estudo de soluções para integração de tecnologias. O DPWS surge como opção de integração para este este tipo de ambiente, e o DSB surge como uma camada alternativa que permite integrar dispositivos de recursos restritos ao ambiente DPWS. Ambos seguem o processo contínuo de maturação, mas apresentam-se como opções viáveis.

O *BluetoothConverter* segue o mesmo caminho de maturação. Para o propósito deste trabalho, acreditamos que o objetivo principal foi atingido, que é o desenvolvimento de uma camada que permita integrar dispositivos Bluetooth a um ambiente de web services. O *BluetoothConverter* foi desenvolvido e mostra-se funcional. Foi possível integrar um serviço Bluetooth simples, de forma que este possa ser invocado por outros dispositivos não Bluetooth, assim como um dispositivo Bluetooth pôde se comunicar com o DSB para invocar outros serviços. Os testes apresentaram resultados satisfatórios, onde o DSB e o *BluetoothConverter* em si não prejudicaram o tempo de resposta, ocorrendo apenas a demora no tempo de resposta por causa da velocidade de transmissão que é natural da tecnologia Bluetooth.

Algumas limitações encontradas durante o desenvolvimento deste trabalho podem ainda servir de inspiração para o desenvolvimento de trabalhos futuros.

- O *BluetoothConverter* neste momento é limitado a tratar apenas uma conexão Bluetooth por vez, isto porque existe uma limitação da tecnologia onde apenas uma conexão pode existir em um determinado instante. Isto implica que não é possível, por exemplo, atender a dois dispositivos Bluetooth simultaneamente, ou atender a uma requisição de um dispositivo Bluetooth e um chamado DSB para outro dispositivo Bluetooth. Desta forma, o estudo de uma alternativa para o tratamento de conexões diferentes é um trabalho a ser avaliado;

O *BluetoothConverter* foi testado com a execução de serviços Bluetooth simples, como serviços de eco que retornavam a mesma mensagem enviada. Uma possibilidade é estudar a integração do *BluetoothConverter* com um dos perfis de serviços Bluetooth, como a utilização de um serviço de impressão;

O *BluetoothConverter* foi desenvolvido utilizando as tecnologias Java, porém com o avanço do desenvolvimento do DSB para outras tecnologias, a conversão do *BluetoothConverter* para estas outras tecnologias também podem ser uma fonte de estudo.

# 8. Referências

PALO, 2008: Palo Alto Research Center, Ubiquotous Computing, 2008, http://www.ubiq.com/hypertext/weiser/UbiHome.html

W3C, 2008a: World Wide Web Consortium, Web Services Activity, 2008, http://www.w3.org/2002/ws

BLUETOOTH SIG, 2008a: Bluetooth SIG, How Bluetooth Technology works, 2008, http://www.bluetooth.com/Bluetooth/Technology/Work

ARAUJO SIQUEIRA, 2008: Gustavo Medeiros Araújo, Frank Siqueira, The Device Service Bus: A solution for Embedded Device Integration throught Web Services, 2008

OPEN GROUP, 2009: Open Group, SOA Source Group, 2009, http://www.opengroup.org/projects/soa-book/

IBM, 2009: IBM, Defining SOA as an Architectural Style, 2009, http://www.ibm.com/developerworks/library/ar-soastyle/

SINGH BRYDON et al, 2004: Inderjeet Singh, Sean Brydon, Greg Murray, Vijay Ramachandran, Thierry Violleau, Beth Stearns, Designing Web Services with the J2EE 1.4 Plataform, 2004

Zigbee, 2009: Zigbee, Zigbee Alliance, 2009, http://www.zigbee.org

JAVA COMMUNITY PROCESS, 2008: Java Community Process, The Java Community Process, 2008, http://jcp.org/en/introduction/faq

MSDN, 2008a: Microsoft, A Technical Introduction to the Devices Profile for Web Services, 2008, http://msdn.microsoft.com/en-us/library/ms996400.aspx

SUN, 2009: SUN, What is an Interface?, 2009, http://java.sun.com/docs/books/tutorial/java/concepts/interface.html

# ANEXO B – CÓDIGO FONTE

```
001 /*
002  *
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027 package br.ufsc.inf.ppgcc.converter.bluetooth;
028
029 import br.ufsc.inf.ppgcc.converter.base.Converter;
030 import br.ufsc.inf.ppgcc.converter.bluetooth.impl.BluetoothHandlerMargeImpl;
031 import br.ufsc.inf.ppgcc.converter.bluetooth.services.impl.BluetoothDSBServicesProviderImpl;
032 import br.ufsc.inf.ppgcc.converter.bluetooth.services.BluetoothDSBServicesProvider;
033 import br.ufsc.inf.ppgcc.converter.entity.ActionConverter;
034 import br.ufsc.inf.ppgcc.search.TunnelSearchRequest;
```

```
035 import br.ufsc.inf.ppgcc.search.TunnelSearchResult;
036 import java.util.concurrent.TimeUnit;
037 import java.util.concurrent.locks.Lock;
038 import java.util.concurrent.locks.ReentrantLock;
039 import org.apache.log4j.Logger;
040
041 /**
042  * Converter responsible for BluetoothCommunication.
043  * @author fabio
044  */
045 public class BluetoothConverter extends Converter {
046
047     private final static Logger LOGGER = Logger.getLogger(BluetoothConverter.class);
048     private BluetoothHandler bluetoothHandler;
049     private BluetoothDSBServicesProvider discoveryService;
050     /**
051      * Currently, the JSR doesn't support multiple stream connections, so,
052      * only one Bluetooth communication can be done per time. This lock says
053      * if a Bluetooth connection is currently in use.
054      */
055     private Lock bluetoothLock;
056
057     /**
058      * Creates a new BluetoothConverter.
059      */
060     public BluetoothConverter() {
061
062         this.bluetoothLock = new ReentrantLock();
063
064         this.setBluetoothHandler(new BluetoothHandlerMargeImpl());
065         this.bluetoothHandler.setBluetoothConverter(this);
066
067         this.discoveryService = new BluetoothDSBServicesProviderImpl();
068         this.discoveryService.setBluetoothConverter(this);
069     }
070
071     @Override
072     public boolean isDeviceAlive() {
073         throw new UnsupportedOperationException("Not supported yet.");
074     }
075
076     /**
077      * Starts the BluetoothConverter.
078      * This will clear the current devices cache and search for devices.
079      */
080     @Override
081     public void startConverter() {
```

```
082         this.bluetoothHandler.clearDevices();
083         this.bluetoothHandler.searchDevices();
084 
085         this.discoveryService.startProvidingService();
086     }
087 
088     /**
089      * Invoques a Bluetooth action.
090      * @param actionConvertor ActionConverter to be invoqued.
091      * @return ActionConverter with the response.
092      */
093     public ActionConverter involkeActionConverter(ActionConverter actionConvertor) {
094 
095         try {
096             if (this.bluetoothLock.tryLock(30, TimeUnit.SECONDS)) {
097                 ActionConverter response = null;
098                 try {
099                     this.discoveryService.stopProvidingService();
100                      response = this.bluetoothHandler.involkeActionConverter(actionConvertor);
101                 } finally {
102                     //this.discoveryService.startProvidingService();
103                     this.bluetoothLock.unlock();
104                     return response;
105                 }
106             }
107         } catch (InterruptedException ex) {
108             LOGGER.error(ex);
109         }
110 
111         actionConvertor.getParameterByName(BluetoothHandlerMargeImpl.BLUETOOTH_PARAMETER_NAME_OUTPUT).
112         return actionConvertor;
113     }
114 
115     @Override
116     public TunnelSearchResult invokeActionThrowTunnel(TunnelSearchResult tunnelSearchResult, String se
117         return super.invokeActionThrowTunnel(tunnelSearchResult, serviceName, actionName);
118     }
119 
120     /**
121      * Search for Services on the DSB.
122      * The response will be sent with a call to onServiceFoundByTunnel();
123      * @param request TunnelSearchRequest with the request info.
124      */
125     public void searchServices(TunnelSearchRequest request) {
126         this.searchDevice(request);
127     }
128 
```

```
129     @Override
130     public void advertiseBridgeAboutDeviceConverterCacheUpdate() {
131         throw new UnsupportedOperationException("Not supported yet.");
132     }
133
134     public void onServiceFoundByTunnel(TunnelSearchResult result) {
135         this.discoveryService.setFoundServices(result);
136     }
137
138     public void onDeviceFoundByTunnel(TunnelSearchResult result) {
139         LOGGER.info("Device discovered: " + result.getDeviceConverter().getDevicename());
140     }
141
142     /**
143      * @param bluetoothHandler the bluetoothHandler to set
144      */
145     public void setBluetoothHandler(BluetoothHandler bluetoothHandler) {
146         this.bluetoothHandler = bluetoothHandler;
147     }
148 }
```

```
01 /*
02  *
03  * $Id$
04  *
05  * The MIT License
06  *
07  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
08  *
09  * Permission is hereby granted, free of charge, to any person obtaining a copy
10  * of this software and associated documentation files (the "Software"), to deal
11  * in the Software without restriction, including without limitation the rights
12  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
13  * copies of the Software, and to permit persons to whom the Software is
14  * furnished to do so, subject to the following conditions:
15  *
16  * The above copyright notice and this permission notice shall be included in
17  * all copies or substantial portions of the Software.
18  *
19  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
20  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
21  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
22  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
23  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
24  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
```

```
25  * THE SOFTWARE.
26  */
27
28 package br.ufsc.inf.ppgcc.converter.bluetooth;
29
30 import br.ufsc.inf.ppgcc.converter.entity.ActionConverter;
31
32 /**
33  * Interface with BluetoothCommunication methods.
34  * @author Fabio Kreusch <fabio at kreusch.com.br>
35  */
36 public interface BluetoothHandler {
37
38     public static final String TYPE_CONVERTOR = BluetoothConverter.class.getSimpleName();
39     public static final String NAMESPACE = "BluetoothDevices";
40     public static final String PREFIX = "BT";
41     public static final String BLUETOOTH_PARAMETER_NAME_INPUT = "BluetoothInputParam";
42     public static final String BLUETOOTH_PARAMETER_NAME_OUTPUT = "BluetoothOutputParam";
43
44     /**
45      * Invokes a Bluetooth action.
46      * @param actionConvertor ActionConverter to be invoqued.
47      * @return ActionConverter with the response.
48      */
49     public ActionConverter involkeActionConverter(ActionConverter actionConvertor);
50
51     /**
52      * Search devices available into area and their services,
53      * and add them to the DSB.
54      */
55     public void searchDevices();
56
57     /**
58      * Removes all devices from the DSB.
59      */
60     public void clearDevices();
61
62     /**
63      * Sets this handler's BluetoothConverter.
64      */
65     public void setBluetoothConverter(BluetoothConverter converter);
66 }
```

```
001 /*
002  *
```

```
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027
028 package br.ufsc.inf.ppgcc.converter.bluetooth;
029
030 import br.ufsc.inf.ppgcc.converter.base.Converter;
031 import br.ufsc.inf.ppgcc.converter.entity.ActionConverter;
032 import br.ufsc.inf.ppgcc.converter.entity.DeviceConverter;
033 import br.ufsc.inf.ppgcc.converter.entity.ParameterConverter;
034 import br.ufsc.inf.ppgcc.converter.entity.ParameterTypeConverter;
035 import br.ufsc.inf.ppgcc.converter.entity.ServiceConverter;
036 import br.ufsc.inf.ppgcc.search.TunnelSearchResult;
037 import java.util.Vector;
038 import org.apache.log4j.Logger;
039 import org.ws4d.java.util.IDGenerator;
040
041 /**
042  *
043  * @author Fabio Kreusch <fabio at kreusch.com.br>
044  */
045 public class FakeConverter extends Converter {
046
047     private final static Logger LOGGER = Logger.getLogger(FakeConverter.class);
048
049     @Override
```

```
050     public boolean isDeviceAlive() {
051         throw new UnsupportedOperationException("Not supported yet.");
052     }
053
054     @Override
055     public void startConverter() {
056         DeviceConverter converter = new DeviceConverter();
057
058         converter.setTypeConvertor(FakeConverter.class.getSimpleName());
059         converter.setDevicename("FakeConverter");
060         converter.setPortType("FakeConverter");
061         converter.setNamespace("FakeConverters");
062         converter.setUuid(IDGenerator.UUID_PREFIX + IDGenerator.getUUID());
063         converter.setPrefix("FAKE");
064
065         converter.setFirmware("FAKE_FIRMWARE");
066         converter.setFriendlyNameLanguage("FAKE_LANGUAGE");
067         converter.setFriendlyNameLocalization("FAKE_LOCALIZATION");
068         converter.setManufactureLanquage("FAKE_LANGUAGE");
069         converter.setManufactureLocalization("FAKE_LOCALIZATION");
070         converter.setModel("FAKE_MODEL");
071         converter.setUrlModel("FAKE_URL_MODEL");
072         Vector v = new Vector();
073         v.add("FAKE_SCOPE");
074         converter.setScopes(v);
075
076         ServiceConverter sc = new ServiceConverter();
077         sc.setName("FAKE_SERVICE");
078         sc.setPortType("FAKE_SERVICE");
079         sc.setNamespace("FakeConverters");
080         sc.setPrefix("FAKE");
081
082         ActionConverter ac = new ActionConverter();
083         ac.setName("FAKE_SERVICE");
084         ac.setOneway(false);
085
086         //Input Parameter Converter
087         ParameterConverter pcIn = new ParameterConverter();
088         pcIn.setDirection(ParameterConverter.INPUT);
089         pcIn.setAttachment(false);
090         pcIn.setComplexType(false);
091         pcIn.setType(ParameterTypeConverter.TYPE_STRING);
092         pcIn.setName(BluetoothHandler.BLUETOOTH_PARAMETER_NAME_INPUT);
093         pcIn.setNameSpace("FakeConverters");
094
095         //Output Parameter Converter
096         ParameterConverter pcOut = new ParameterConverter();
```

```
097         pcOut.setDirection(ParameterConverter.OUTPUT);
098         pcOut.setAttachment(false);
099         pcOut.setComplexType(true);
100         pcOut.setType(ParameterTypeConverter.TYPE_STRING);
101         pcOut.setName(BluetoothHandler.BLUETOOTH_PARAMETER_NAME_OUTPUT);
102         pcOut.setNameSpace("FakeConverters");
103 
104         ac.addParameter(pcIn);
105         ac.addParameter(pcOut);
106         sc.addAction(ac);
107         converter.addService(sc);
108 
109         this.addDeviceConverter(converter);
110         LOGGER.info("FakeDevice added");
111     }
112 
113     @Override
114     public ActionConverter involkeActionConverter(ActionConverter action) {
115         action.getParameterByName(BluetoothHandler.BLUETOOTH_PARAMETER_NAME_OUTPUT).setValue("FAKE_RES
116         return action;
117     }
118 
119     @Override
120     public void advertiseBridgeAboutDeviceConverterCacheUpdate() {
121         throw new UnsupportedOperationException("Not supported yet.");
122     }
123 
124     public void onServiceFoundByTunnel(TunnelSearchResult arg0) {
125         throw new UnsupportedOperationException("Not supported yet.");
126     }
127 
128     public void onDeviceFoundByTunnel(TunnelSearchResult arg0) {
129         throw new UnsupportedOperationException("Not supported yet.");
130     }
131 
132 }
```

```
001 /*
002  *
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
```

```
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027 package br.ufsc.inf.ppgcc.converter.bluetooth.impl;
028
029 import br.ufsc.inf.ppgcc.converter.bluetooth.BluetoothConverter;
030 import br.ufsc.inf.ppgcc.converter.bluetooth.BluetoothHandler;
031 import br.ufsc.inf.ppgcc.converter.bluetooth.impl.communication.ActionConverterCommunicationListener;
032 import br.ufsc.inf.ppgcc.converter.bluetooth.impl.communication.ActionConverterCommunicationListenerFa
033 import br.ufsc.inf.ppgcc.converter.entity.ActionConverter;
034 import br.ufsc.inf.ppgcc.converter.entity.DeviceConverter;
035 import br.ufsc.inf.ppgcc.converter.entity.ParameterConverter;
036 import br.ufsc.inf.ppgcc.converter.entity.ParameterTypeConverter;
037 import br.ufsc.inf.ppgcc.converter.entity.ServiceConverter;
038 import java.io.IOException;
039 import java.util.ArrayList;
040 import java.util.Enumeration;
041 import java.util.HashMap;
042 import java.util.Iterator;
043 import java.util.List;
044 import java.util.Map;
045 import java.util.Set;
046 import javax.bluetooth.BluetoothStateException;
047 import javax.bluetooth.DataElement;
048 import javax.bluetooth.DeviceClass;
049 import javax.bluetooth.RemoteDevice;
050 import javax.bluetooth.ServiceRecord;
051 import javax.bluetooth.UUID;
052 import net.java.dev.marge.inquiry.DeviceDiscoverer;
053 import net.java.dev.marge.inquiry.InquiryListener;
054 import net.java.dev.marge.inquiry.ServiceDiscoverer;
055 import net.java.dev.marge.inquiry.ServiceSearchListener;
```

```
056 import org.apache.log4j.Logger;
057 import org.ws4d.java.util.IDGenerator;
058
059 /**
060  * Implementation of BluetoothHandler.
061  * This implementation uses the Marge library to control the Bluetooth Communication.
062  * @author Fabio Kreusch <fabio at kreusch.com.br>
063  */
064 public class BluetoothHandlerMargeImpl implements BluetoothHandler, InquiryListener {
065
066     private final static Logger LOGGER = Logger.getLogger(BluetoothHandlerMargeImpl.class);
067     private static int serviceCode = 1;
068     private BluetoothConverter bluetoothConverter;
069     protected Map<DeviceConverter, RemoteDevice> convertersToRemoteDevices;
070     protected Map<String, DeviceConverter> actionsToDevices;
071     protected Map<String, ServiceRecord> actionsToServices;
072     protected ServiceDiscovererObserver serviceDiscovererObserver;
073
074     public BluetoothHandlerMargeImpl() {
075         this.convertersToRemoteDevices = new HashMap<DeviceConverter, RemoteDevice>();
076         this.actionsToDevices = new HashMap<String, DeviceConverter>();
077         this.actionsToServices = new HashMap<String, ServiceRecord>();
078     }
079
080     /**
081      * Searches for BluetoothDevices and add them to the BluetoothConverter.
082      * This is an asynchronous process. The searchDevices() will start a thread to
083      * inquiry for Bluetooth Services. When the inquiry is complete, the inquiryComplete method is
084      * called to add the found services to the BluetoothConverter.
085      */
086     public void searchDevices() {
087         DeviceDiscovererThread thread = new DeviceDiscovererThread();
088         thread.start();
089     }
090
091     /**
092      * Clears all devices from the DSB.
093      */
094     public synchronized void clearDevices() {
095         Set<DeviceConverter> converters = this.convertersToRemoteDevices.keySet();
096         for (DeviceConverter converter : converters) {
097             this.bluetoothConverter.removeDeviceConverter(converter);
098         }
099         this.convertersToRemoteDevices.clear();
100         this.actionsToDevices.clear();
101         this.actionsToServices.clear();
102         LOGGER.info("Bluetooth DeviceConverters cleared.");
```

```
103     }
104
105     /**
106      * @param bluetoothConverter the bluetoothConverter to set
107      */
108     public void setBluetoothConverter(BluetoothConverter bluetoothConverter) {
109         this.bluetoothConverter = bluetoothConverter;
110     }
111
112     /*
113      * InquiryListener methods.
114      */
115     /**
116      * Called by Marge when a RemoteDevice is found.
117      * @param remoteDevice RemoteDevice
118      * @param deviceClass DeviceClass
119      */
120     public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass) {
121         LOGGER.info("Bluetooth Device discovered: " + remoteDevice.getBluetoothAddress());
122     }
123
124     /**
125      * Called by Marge when the inquiry process is completed.
126      * This will add all RemoteDevices found to the observer and start the observer that
127      * will search for services.
128      * @param remoteDevices RemoteDevice[] A list with all RemoteDevices found.
129      */
130     public void inquiryCompleted(RemoteDevice[] remoteDevices) {
131         this.serviceDiscovererObserver = new ServiceDiscovererObserver();
132         this.serviceDiscovererObserver.setRemoteDevices(remoteDevices);
133         this.serviceDiscovererObserver.startServicesSearch();
134     }
135
136     /**
137      * Called by Marge when an error ocours during a device inquiry process.
138      */
139     public void inquiryError() {
140         LOGGER.info("Error during bluetooth devices inquiry.");
141     }
142
143     /**
144      * Called by the ServiceDiscovererObserver when the search for device's services is completed.
145      * Maps the RemoteDevice and his ServiceRecords to a DeviceConverter and adds it to the Bluetooth(
146      * @param devicesAndServices a map with devices and services.
147      */
148     protected void servicesSearchCompleted(Map<RemoteDevice, List<ServiceRecord>> devicesAndServices)
149         LOGGER.info("Starting BluetoothRecords conversion to DeviceConverters (" + devicesAndServices.
```

```
150
151         try {
152             for (RemoteDevice remoteDevice : devicesAndServices.keySet()) {
153                 DeviceConverter converter = this.convertRemoteDeviceToDeviceConverter(remoteDevice);
154                 this.convertersToRemoteDevices.put(converter, remoteDevice);
155                 for (ServiceRecord service : devicesAndServices.get(remoteDevice)) {
156                     //Configuring the Service Converter
157                     ServiceConverter serviceConverter = this.convertServiceRecordToServiceConverter(se
158
159                     //Adds ServiceConverter to DeviceConverter
160                     converter.addService(serviceConverter);
161
162                     //maps the ActionConverters to a DeviceConverter
163                     Enumeration e = serviceConverter.getActions().elements();
164                     while (e.hasMoreElements()) {
165                         this.actionsToDevices.put(((ActionConverter) e.nextElement()).getName(), conve
166                     }
167                 }
168
169                 this.bluetoothConverter.addDeviceConverter(converter);
170                 LOGGER.info("DeviceConverter added: " + converter.getDevicename());
171             }
172         } catch (IOException ex) {
173             LOGGER.error(ex.getMessage(), ex);
174         }
175     }
176     /*
177      * End of InquiryListener methods.
178      */
179
180     /**
181      * Converts a RemoteDevice bluetooth device to a DeviceConverter DSB device.
182      * @param RemoteDevice
183      * @return DeviceConverter
184      * @throws java.io.IOException
185      */
186     protected DeviceConverter convertRemoteDeviceToDeviceConverter(RemoteDevice remoteDev) throws IOEx
187         DeviceConverter converter = new DeviceConverter();
188
189         String devName = remoteDev.getFriendlyName(false) != null ? remoteDev.getFriendlyName(false) :
190
191         converter.setTypeConvertor(TYPE_CONVERTOR);
192         converter.setDevicename(devName);
193         converter.setPortType(devName);
194         converter.setNamespace(NAMESPACE);
195         converter.setUuid(IDGenerator.UUID_PREFIX + IDGenerator.getUUID());
196         converter.setPrefix(PREFIX);
```

```
197
198         converter.setFriendlyNameLocalization(devName);
199         converter.setManufactureLocalization(devName);
200         converter.setModel("BluetoothDevice");
201         converter.setFriendlyNameLanguage("Unknown");
202         converter.setManufactureLanquage("Unknown");
203
204         return converter;
205     }
206
207     /**
208      * Converts a ServiceRecord bluetooth service to a ServiceConverter DSB service with its ActionCor
209      * and ParameterConverters.
210      * @param ServiceRecord
211      * @return ServiceConverter
212      * @throws java.io.IOException
213      */
214     protected ServiceConverter convertServiceRecordToServiceConverter(ServiceRecord service) throws IO
215         ServiceConverter sc = new ServiceConverter();
216
217         //Service Converter
218
219         DataElement serviceNameData = service.getAttributeValue(0x0100);
220         String serviceName = null;
221         if (serviceNameData != null) {
222             serviceName = (String) serviceNameData.getValue();
223         } else {
224             serviceName = "UnknownService";
225         }
226
227         sc.setName(serviceName);
228         sc.setPortType(serviceName);
229         sc.setNamespace(NAMESPACE);
230         sc.setPrefix(PREFIX);
231
232         //Action Converter
233         ActionConverter ac = new ActionConverter();
234         ac.setName(serviceName);
235         ac.setOneway(false);
236
237         //Input Parameter Converter
238         ParameterConverter pcIn = new ParameterConverter();
239         pcIn.setDirection(ParameterConverter.INPUT);
240         pcIn.setAttachment(false);
241         pcIn.setComplexType(false);
242         pcIn.setType(ParameterTypeConverter.TYPE_STRING);
243         pcIn.setName(BLUETOOTH_PARAMETER_NAME_INPUT);
```

```
244         pcIn.setNameSpace(NAMESPACE);
245 
246         //Output Parameter Converter
247         ParameterConverter pcOut = new ParameterConverter();
248         pcOut.setDirection(ParameterConverter.OUTPUT);
249         pcOut.setAttachment(false);
250         pcOut.setComplexType(true);
251         pcOut.setType(ParameterTypeConverter.TYPE_STRING);
252         pcOut.setName(BLUETOOTH_PARAMETER_NAME_OUTPUT);
253         pcOut.setNameSpace(NAMESPACE);
254 
255         //adds parameter and action
256         ac.addParameter(pcIn);
257         ac.addParameter(pcOut);
258         sc.addAction(ac);
259 
260         //maps the ActionConverter to the ServiceRecord.
261         this.actionsToServices.put(ac.getName(), service);
262 
263         return sc;
264     }
265 
266     /*
267      * Communication methods
268      */
269     /**
270      * Invokes a Bluetooth action.
271      * @param actionConvertor ActionConverter to be invoqued.
272      * @return ActionConverter with the response.
273      */
274     public ActionConverter involkeActionConverter(ActionConverter actionConvertor) {
275         ServiceRecord service = this.actionsToServices.get(actionConvertor.getName());
276 
277         if (service == null) {
278             LOGGER.error("No ServiceRecord found for ActionConverter " + actionConvertor.getName());
279             return actionConvertor;
280         }
281 
282         ActionConverterCommunicationListener listener = ActionConverterCommunicationListenerFactory.cr
283         if (listener != null) {
284             synchronized (listener) {
285                 try {
286                     listener.sendMessage();
287                     listener.wait(60000);
288                 } catch (InterruptedException ex) {
289                     LOGGER.error(ex);
290                 }
```

```
291             }
292         }
293 
294         //if no response is found, remove deviceConverter
295         if (listener == null || !listener.hasResponse()) {
296             DeviceConverter toRemove = this.actionsToDevices.get(actionConvertor.getName());
297             this.bluetoothConverter.removeDeviceConverter(toRemove);
298             this.actionsToDevices.remove(actionConvertor.getName());
299             this.actionsToServices.remove(actionConvertor.getName());
300             this.convertersToRemoteDevices.remove(actionConvertor);
301 
302             actionConvertor.getParameterByName(BluetoothHandlerMargeImpl.BLUETOOTH_PARAMETER_NAME_OUTP
303 
304             LOGGER.info("Returning empty involkeActionConverter call");
305             return actionConvertor;
306         }
307 
308         LOGGER.info("Returning involkeActionConverter call");
309         return listener.getActionConverter();
310     }
311 
312     /**
313      * A Thread that inquiry for devices.
314      */
315     class DeviceDiscovererThread extends Thread {
316 
317         @Override
318         public void run() {
319             try {
320                 LOGGER.info("Starting Device discovery");
321                 DeviceDiscoverer.getInstance().startInquiryGIAC(BluetoothHandlerMargeImpl.this);
322             } catch (BluetoothStateException ex) {
323                 LOGGER.error(ex.getMessage(), ex);
324             }
325         }
326     }
327 
328     /**
329      * A Thread that observes a group of ServiceDiscovererThreads and
330      * notice when all of them have finished.
331      */
332     class ServiceDiscovererObserver implements ServiceSearchListener {
333 
334         protected ArrayList<RemoteDevice> remoteDevices;
335         protected Map<RemoteDevice, List<ServiceRecord>> devicesAndServices = new HashMap<RemoteDevice
336         protected final Object serviceDiscoveryLock = new Object();
337 
```

```
338         /**
339          * Sets the RemoteDevice[] list to search for services.
340          * @param remoteDevices the RemoteDevice[] list
341          */
342         protected void setRemoteDevices(RemoteDevice[] remoteDevices) {
343             if (remoteDevices != null) {
344                 this.remoteDevices = new ArrayList<RemoteDevice>();
345                 for (RemoteDevice device : remoteDevices) {
346                     this.remoteDevices.add(device);
347                 }
348                 LOGGER.info(this.remoteDevices.size() + " RemoteDevices added to Observer.");
349             }
350         }
351 
352         /**
353          * Starts the Services search for each RemoteDevice.
354          */
355         protected void startServicesSearch() {
356             synchronized (this.serviceDiscoveryLock) {
357                 Iterator<RemoteDevice> it = this.remoteDevices.iterator();
358                 List<UUID[]> uuids = BluetoothHandlerUtils.getBluetoothServicesUUID();
359                 while (it.hasNext()) {
360                     RemoteDevice device = it.next();
361                     //search for each service
362                     for (UUID[] service : uuids) {
363                         try {
364                             ServiceDiscoverer.getInstance().startSearch(service, device, this);
365                             //Waits for the service discovery to end
366                             serviceDiscoveryLock.wait();
367                         } catch (BluetoothStateException ex) {
368                             LOGGER.error(ex);
369                         } catch (InterruptedException ex) {
370                             LOGGER.error(ex);
371                         }
372                     }
373                 }
374                 BluetoothHandlerMargeImpl.this.servicesSearchCompleted(devicesAndServices);
375             }
376         }
377 
378         /**
379          * This is called by Marge after a Services search is completed for a RemoteDevice.
380          * It will store the remoteDevice with the services found for it.
381          * @param remoteDevice the RemoteDevice whose search was completed.
382          * @param services the RemoteDevice found services.
383          */
384         public void serviceSearchCompleted(RemoteDevice remoteDevice, ServiceRecord[] services) {
```

```
385             synchronized (this.serviceDiscoveryLock) {
386                 try {
387                     if (services.length > 0 && services[0] != null) {
388                         if (this.devicesAndServices.get(remoteDevice) == null) {
389                             List<ServiceRecord> servicesList = new ArrayList<ServiceRecord>();
390                             servicesList.add(services[0]);
391                             this.devicesAndServices.put(remoteDevice, servicesList);
392                         } else {
393                             this.devicesAndServices.get(remoteDevice).add(services[0]);
394                         }
395                     }
396 
397                     LOGGER.info("Service discovery completed for " + remoteDevice.getFriendlyName(fals
398                 } catch (IOException ex) {
399                     LOGGER.error(ex);
400                 } finally {
401                     this.serviceDiscoveryLock.notify();
402                 }
403             }
404         }
405 
406         /**
407          * This is called by Marge if for some reason a RemoteDevice is not anymore reachable.
408          */
409         public void deviceNotReachable() {
410             synchronized (this.serviceDiscoveryLock) {
411                 try {
412                     LOGGER.info("Device not reachable.");
413                 } finally {
414                     this.serviceDiscoveryLock.notify();
415                 }
416             }
417         }
418 
419         /**
420          * This is called by Marge if an error ocours during ServicesSearch.
421          */
422         public void serviceSearchError() {
423             synchronized (this.serviceDiscoveryLock) {
424                 try {
425                     LOGGER.info("Error during service search for device.");
426                 } finally {
427                     this.serviceDiscoveryLock.notify();
428                 }
429             }
430         }
431     }
```

```
432 }
```

```
001 /*
002  *
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027 package br.ufsc.inf.ppgcc.converter.bluetooth.impl;
028
029 import java.io.ByteArrayInputStream;
030 import java.io.ByteArrayOutputStream;
031 import java.io.IOException;
032 import java.io.ObjectInputStream;
033 import java.io.ObjectOutputStream;
034 import java.util.ArrayList;
035 import java.util.Enumeration;
036 import java.util.List;
037 import java.util.Properties;
038 import javax.bluetooth.UUID;
039 import org.apache.log4j.Logger;
040
041 /**
042  * Contains utility methods for the BluetoothHandler.
043  * @author Fabio Kreusch <fabio at kreusch.com.br>
```

```
044  */
045 public class BluetoothHandlerUtils {
046 
047     private final static Logger LOGGER = Logger.getLogger(BluetoothHandlerUtils.class);
048     private final static String PROPERTIES_PATH = "br/ufsc/inf/ppgcc/converter/bluetooth/impl/bluetoo
049 
050     /**
051      * Returns a list of Bluetooth services UUIDs to search.
052      * @return the services to search UUID[] List.
053      */
054     public static List<UUID[]> getBluetoothServicesUUID() {
055         return loadUUIDsFromProperties(BluetoothHandlerUtils.loadPropertiesFile());
056     }
057 
058     /**
059      * Extracts the UUIDs to search into the properties file.
060      * @param props the properties file
061      * @return
062      */
063     protected static List<UUID[]> loadUUIDsFromProperties(Properties props) {
064         List<UUID> uuids = new ArrayList();
065 
066         Enumeration enumeration = props.elements();
067 
068         while (enumeration.hasMoreElements()) {
069             String value = (String) enumeration.nextElement();
070             //if starts with an 's' is assumed to be a shortUUID
071             if (value.startsWith("s")) {
072                 value = value.replaceAll("s", "");
073                 uuids.add(new UUID(value, true));
074             } else {
075                 uuids.add(new UUID(value, false));
076             }
077         }
078 
079         return formatted(uuids);
080     }
081 
082     /**
083      * Loads the Properties file.
084      * @return Properties
085      */
086     protected static Properties loadPropertiesFile() {
087         Properties servicesToSearch = new Properties();
088         try {
089             servicesToSearch.load(ClassLoader.getSystemResourceAsStream(BluetoothHandlerUtils.PROPERTI
090         } catch (IOException ex) {
```

```
091             LOGGER.error(ex.getMessage(), ex);
092         }
093         return servicesToSearch;
094     }
095 
096     /**
097      * Formats a UUID List to the format required for ServiceDiscovery in Bluetooth (a List<UUID[]>).
098      * @param uuids the UUID List
099      * @return a List<UUID[]>
100      */
101     protected static List<UUID[]> formatted(List<UUID> uuids) {
102         List<UUID[]> formatted = new ArrayList<UUID[]>();
103 
104         for (UUID uuid : uuids) {
105             UUID[] array = {uuid};
106             formatted.add(array);
107         }
108 
109         return formatted;
110     }
111 
112     /**
113      * Converts a object to a byte array.
114      * @param obj the object to convert
115      * @return the byte array
116      * @throws java.io.IOException
117      */
118     /*public static byte[] getBytes(Object obj) throws java.io.IOException {
119         ByteArrayOutputStream bos = new ByteArrayOutputStream();
120         ObjectOutputStream oos = new ObjectOutputStream(bos);
121         oos.writeObject(obj);
122         oos.flush();
123         oos.close();
124         bos.close();
125         byte[] data = bos.toByteArray();
126         return data;
127     }*/
128 
129     /**
130      * Converts a byte array back to a object.
131      * @param bytes the byte array to convert
132      * @return the converted object
133      * @throws java.io.IOException
134      * @throws java.lang.ClassNotFoundException
135      */
136     /*public static Object getObject(byte[] bytes) throws IOException, ClassNotFoundException {
137         ByteArrayInputStream bis = new ByteArrayInputStream(bytes);
```

```
138         ObjectInputStream ois = new ObjectInputStream(bis);
139         Object object = ois.readObject();
140         ois.close();
141         bis.close();
142         return object;
143     }*/
144 }
```

```
001 /*
002  *
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027 package br.ufsc.inf.ppgcc.converter.bluetooth.impl.communication;
028
029 import br.ufsc.inf.ppgcc.converter.bluetooth.impl.*;
030 import br.ufsc.inf.ppgcc.converter.entity.ActionConverter;
031 import br.ufsc.inf.ppgcc.converter.entity.ParameterConverter;
032 import java.io.IOException;
033 import javax.bluetooth.BluetoothStateException;
034 import javax.bluetooth.ServiceRecord;
035 import net.java.dev.marge.communication.CommunicationListener;
036 import net.java.dev.marge.entity.ClientDevice;
037 import net.java.dev.marge.entity.config.ClientConfiguration;
```

```
038 import net.java.dev.marge.factory.CommunicationFactory;
039 import org.apache.log4j.Logger;
040
041 /**
042  * This class is a listener that implements a CommunicationListerner
043  * for an ActionConverter.
044  * @author Fabio Kreusch <fabio at kreusch.com.br>
045  */
046 public class ActionConverterCommunicationListener implements CommunicationListener {
047
048     private final static Logger LOGGER = Logger.getLogger(ActionConverterCommunicationListener.class);
049     private ActionConverter actionConverter;
050     private ServiceRecord serviceRecord;
051     private ClientConfiguration clientConfiguration;
052     private ClientDevice clientDevice;
053     private CommunicationFactory communicationFactory;
054     private boolean hasResponse = false;
055     private boolean isConnected = false;
056
057     /**
058      * Default Constructor to create a new Bluetooth CommunicationListener.
059      * @param action The ActionConverte> this listener will responds to.
060      */
061     public ActionConverterCommunicationListener(ActionConverter action, ServiceRecord service, Communi
062         this.actionConverter = action;
063         this.serviceRecord = service;
064         this.communicationFactory = commFactory;
065     }
066
067     /**
068      * Returns if this listener have already a response.
069      * @return true if have already a response.
070      */
071     public boolean hasResponse() {
072         return hasResponse;
073     }
074
075     /**
076      * Returns if this listener is connected to a device.
077      * @return true if is connected to a device.
078      */
079     public boolean isConnected() {
080         return isConnected;
081     }
082
083     public void startConnection() {
084         if (!this.isConnected) {
```

```
085            LOGGER.info("Trying to communicate with " + this.getActionConverter().getName());
086
087            try {
088                clientConfiguration = new ClientConfiguration(serviceRecord, this);
089                clientDevice = this.communicationFactory.connectToServer(clientConfiguration);
090
091                if (clientDevice != null) {
092                    clientDevice.startListening();
093                    this.isConnected = true;
094
095                    LOGGER.info("Connected with " + this.getActionConverter().getName() + ", URL: " +
096                }
097            } catch (IOException ex) {
098                LOGGER.error(ex.getMessage(), ex);
099            }
100        } else {
101            LOGGER.info("Already connected to " + this.getActionConverter().getName());
102        }
103    }
104
105    /**
106     * Closes a connection with a device.
107     */
108    public void closeConnection() {
109        this.clientDevice.stopListenning();
110        this.clientDevice.close();
111
112        this.isConnected = false;
113        try {
114            LOGGER.info("Connection closed with " + this.clientDevice.getBluetoothAddress());
115        } catch (BluetoothStateException ex) {
116            LOGGER.error(ex.getMessage(), ex);
117        }
118    }
119
120    /**
121     * Sends a message for the Bluetooth service.
122     */
123    public void sendMessage() {
124        ParameterConverter parameter = getActionConverter().getParameterByName(BluetoothHandlerMargeIm
125        clientDevice.send(parameter.getValue().toString().getBytes());
126
127        LOGGER.info("Message sent to server: " + parameter.getValue().toString());
128    }
129
130    /**
131     * When a Bluetooth message is received, this method is called.
```

```
132      * Then, it stores the received message into the <ActionConverter>.
133      * @param message
134      */
135     public void receiveMessage(byte[] message) {
136         synchronized (this) {
137             try {
138                 this.getActionConverter().getParameterByName(BluetoothHandlerMargeImpl.BLUETOOTH_PARAM
139                 this.hasResponse = true;
140 
141                 LOGGER.info("Bluetooth response received: " + new String(message));
142             } finally {
143                 this.closeConnection();
144                 this.notify();
145             }
146         }
147     }
148 
149     public void errorOnReceiving(IOException ex) {
150         synchronized (this) {
151             try {
152                 LOGGER.error(ex.getMessage(), ex);
153             } finally {
154                 this.closeConnection();
155                 this.notify();
156             }
157         }
158     }
159 
160     public void errorOnSending(IOException ex) {
161         synchronized (this) {
162             try {
163                 LOGGER.error(ex.getMessage(), ex);
164             } finally {
165                 this.closeConnection();
166                 this.notify();
167             }
168         }
169     }
170 
171     /**
172      * @return the actionConverter
173      */
174     public ActionConverter getActionConverter() {
175         return actionConverter;
176     }
177 }
```

```
01 /*
02  *
03  * $Id$
04  *
05  * The MIT License
06  *
07  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
08  *
09  * Permission is hereby granted, free of charge, to any person obtaining a copy
10  * of this software and associated documentation files (the "Software"), to deal
11  * in the Software without restriction, including without limitation the rights
12  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
13  * copies of the Software, and to permit persons to whom the Software is
14  * furnished to do so, subject to the following conditions:
15  *
16  * The above copyright notice and this permission notice shall be included in
17  * all copies or substantial portions of the Software.
18  *
19  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
20  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
21  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
22  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
23  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
24  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
25  * THE SOFTWARE.
26  */
27 package br.ufsc.inf.ppgcc.converter.bluetooth.impl.communication;
28
29 import br.ufsc.inf.ppgcc.converter.entity.ActionConverter;
30 import javax.bluetooth.ServiceRecord;
31 import net.java.dev.marge.factory.RFCOMMCommunicationFactory;
32 import org.apache.log4j.Logger;
33
34 /**
35  * * A Factory to create a new ActionConverterCommunicationListener object to a BluetoothService.
36  * @author Fabio Kreusch <fabio at kreusch.com.br>
37  */
38 public class ActionConverterCommunicationListenerFactory {
39
40     private final static Logger LOGGER = Logger.getLogger(ActionConverterCommunicationListenerFactory.c
41     private static RFCOMMCommunicationFactory communicationFactory;
42
43     /**
44      * Creates a new ActionConverterCommunicationListener object for
45      * a ActionConverter and the corresponding ServiceRecord.
```

```
46      * @param converter the ActionConverter
47      * @param service the ServiceRecord corresponding to the ActionConverter
48      * @return a new ActionConverterCommunicationListener if a connection to the service was possible.
49      */
50     public static ActionConverterCommunicationListener createNewCommunicationListener(ActionConverter c
51 
52         if(communicationFactory == null) {
53             communicationFactory = new RFCOMMCommunicationFactory();
54         }
55 
56         ActionConverterCommunicationListener listener = new ActionConverterCommunicationListener(conve
57         listener.startConnection();
58 
59         if (listener.isConnected()) {
60             return listener;
61         } else {
62             return null;
63         }
64     }
65 }
```

```
01 /*
02  *
03  * $Id$
04  *
05  * The MIT License
06  *
07  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
08  *
09  * Permission is hereby granted, free of charge, to any person obtaining a copy
10  * of this software and associated documentation files (the "Software"), to deal
11  * in the Software without restriction, including without limitation the rights
12  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
13  * copies of the Software, and to permit persons to whom the Software is
14  * furnished to do so, subject to the following conditions:
15  *
16  * The above copyright notice and this permission notice shall be included in
17  * all copies or substantial portions of the Software.
18  *
19  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
20  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
21  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
22  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
23  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
24  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
```

```
25  * THE SOFTWARE.
26  */
27 package br.ufsc.inf.ppgcc.converter.bluetooth.services;
28
29 import br.ufsc.inf.ppgcc.converter.bluetooth.BluetoothConverter;
30 import br.ufsc.inf.ppgcc.search.TunnelSearchResult;
31 import javax.bluetooth.UUID;
32
33 /**
34  * The implementation of this interface should provide a BluetoothService to search
35  * for DSB services.
36  * @author Fabio Kreusch <fabio at kreusch.com.br>
37  */
38 public interface BluetoothDSBServicesProvider {
39
40     public static final String SERVICE_NAME = "DSBDiscoveryService";
41     public static final UUID SERVICE_UUID = new UUID("EFA8637497899AE", false);
42
43     /*public static final String PARAM_UUID = "requestor_uuid";
44     public static final String PARAM_DEVICE_NAMESPACE = "device_namespace";
45     public static final String PARAM_DEVICE_PORTTYPE = "device_porttype";
46     public static final String PARAM_SERVICE_NAMESPACE = "service_namespace";
47     public static final String PARAM_SERVICE_PORTTYPE = "service_porttype";
48     public static final String RESULT_SERVICES = "services";
49     public static final String RESULT_ACTIONS = "actions";*/
50
51     /**
52      * Starts providing the DiscoveryService for Bluetooth devices.
53      */
54     public void startProvidingService();
55
56     /**
57      * Stops providing the DiscoveryService for Bluetooth devices.
58      */
59     public void stopProvidingService();
60
61     /**
62      * Sets the found results. This is called by BluetoothConverter.onServiceFoundByTunnel
63      * after the DSB finds services.
64      * @param result the found results.
65      */
66     public void setFoundServices(TunnelSearchResult result);
67
68     /**
69      * Sets the BluetoothConverter responsible for this discovery service.
70      * @param bluetoothConverter the BluetoothConverter
71      */
```

```
72     public void setBluetoothConverter(BluetoothConverter bluetoothConverter);
73 }
```

```
001 /*
002  *
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027 package br.ufsc.inf.ppgcc.converter.bluetooth.services.impl;
028
029 import br.ufsc.inf.ppgcc.converter.bluetooth.BluetoothConverter;
030 import br.ufsc.inf.ppgcc.converter.bluetooth.services.messages.BluetoothTunnelSearchRequest;
031 import br.ufsc.inf.ppgcc.converter.bluetooth.services.BluetoothDSBServicesProvider;
032 import br.ufsc.inf.ppgcc.converter.bluetooth.services.messages.BluetoothTunnelSearchResult;
033 import br.ufsc.inf.ppgcc.converter.entity.ParameterConverter;
034 import br.ufsc.inf.ppgcc.search.TunnelSearchRequest;
035 import br.ufsc.inf.ppgcc.search.TunnelSearchResult;
036 import java.io.IOException;
037 import java.io.Reader;
038 import java.io.StringReader;
039 import java.util.ArrayList;
040 import java.util.HashMap;
041 import java.util.Iterator;
042 import java.util.List;
```

```
043 import java.util.Map;
044 import java.util.Set;
045 import java.util.Vector;
046 import javax.bluetooth.RemoteDevice;
047 import javax.bluetooth.UUID;
048 import net.java.dev.marge.communication.CommunicationListener;
049 import net.java.dev.marge.communication.ConnectionListener;
050 import net.java.dev.marge.entity.ServerDevice;
051 import net.java.dev.marge.entity.config.ServerConfiguration;
052 import net.java.dev.marge.factory.RFCOMMCommunicationFactory;
053 import org.apache.log4j.Logger;
054 import org.jdom.Document;
055 import org.jdom.Element;
056 import org.jdom.JDOMException;
057 import org.jdom.input.SAXBuilder;
058 import org.jdom.output.XMLOutputter;
059
060 /**
061  * This class provides a BluetoothService to search for DSB services.
062  * @author Fabio Kreusch <fabio at kreusch.com.br>
063  */
064 public class BluetoothDSBServicesProviderImpl implements BluetoothDSBServicesProvider, CommunicationLi
065
066     protected BluetoothConverter bluetoothConverter;
067     protected final static Logger LOGGER = Logger.getLogger(BluetoothDSBServicesProviderImpl.class);
068     protected ServerDevice serverDevice;
069     protected List<RemoteDevice> clientDevices = new ArrayList<RemoteDevice>();
070     private ServiceThread serviceThread;
071     protected Map<TunnelSearchRequest, TunnelSearchResult> requestCache;
072     protected Map<TunnelSearchRequest, String> requestUUIDs;
073     protected Map<String, TunnelSearchResult> UUIDSresult;
074
075     /**
076      * Starts this service and waits for connections.
077      */
078     public void startProvidingService() {
079         if (this.serviceThread == null) {
080
081             this.requestCache = new HashMap<TunnelSearchRequest, TunnelSearchResult>();
082             this.requestUUIDs = new HashMap<TunnelSearchRequest, String>();
083             this.UUIDSresult = new HashMap<String, TunnelSearchResult>();
084
085             this.serviceThread = new ServiceThread();
086             this.serviceThread.start();
087             LOGGER.info("BluetoothService '" + SERVICE_NAME + " : " + SERVICE_UUID + "' is now waiting
088         } else {
089             LOGGER.info("BluetoothService '" + SERVICE_NAME + " : " + SERVICE_UUID + "' already starte
```

```
090         }
091     }
092
093     /**
094      * Stops the service.
095      */
096     public void stopProvidingService() {
097         if (this.serverDevice != null) {
098             this.serverDevice.stopListenning();
099             this.serverDevice.close();
100             this.serverDevice = null;
101             this.clientDevices.clear();
102             LOGGER.info("Stoped " + SERVICE_NAME);
103         }
104
105         if ((this.serviceThread != null)) {
106             this.serviceThread.running = false;
107             this.serviceThread.interrupt();
108
109             try {
110                 this.serviceThread.join();
111             } catch (InterruptedException ex) {
112                 LOGGER.debug(ex);
113             }
114
115             this.serviceThread = null;
116             LOGGER.info("Stoped " + SERVICE_NAME + " thread");
117         } else {
118             LOGGER.info("No active " + SERVICE_NAME);
119         }
120     }
121
122     /**
123      * Sets the found results. This is called by BluetoothConverter.onServiceFoundByTunnel
124      * after the DSB finds services.
125      * @param result the found results.
126      */
127     public void setFoundServices(TunnelSearchResult result) {
128         this.requestCache.put(result.getTunnelSearchRequest(), result);
129
130         LOGGER.info("Result added to cache");
131     }
132
133     /**
134      * @param bluetoothConverter the bluetoothConverter to set
135      */
136     public void setBluetoothConverter(BluetoothConverter bluetoothConverter) {
```

```
137            this.bluetoothConverter = bluetoothConverter;
138        }
139
140        public void connectionEstablished(ServerDevice serverDevice, RemoteDevice remoteDevice) {
141            this.serverDevice = serverDevice;
142            this.serverDevice.setEnableBroadcast(true);
143            this.serverDevice.startListening();
144
145            LOGGER.info("Connection established with " + remoteDevice.getBluetoothAddress());
146        }
147
148        public void initialisationSucessful() {
149            LOGGER.info("BluetoothDSBService initialized");
150        }
151
152        /**
153         * Receives and processes the Bluetooth message. This is called by Marge when a new message is rec
154         * The message is expected as a BluetoothTunnelSearchRequest object, if
155         * this is a call to the discovery service, or a BluetoothTunnelSearchResult, if this is a call to
156         * service.
157         * @param msg The message in bytes format
158         */
159        public void receiveMessage(byte[] bytesMsg) {
160
161            LOGGER.info("Message received: " + new String(bytesMsg));
162
163            try {
164                if (this.isSearchRequest(bytesMsg)) {
165                    BluetoothTunnelSearchRequest request = this.extractSearchRequest(bytesMsg);
166                    this.processDiscoveryRequest(request);
167                } else if (this.isInvokeRequest(bytesMsg)) {
168                    BluetoothTunnelSearchResult result = this.extractInvokeRequest(bytesMsg);
169                    this.processInvokeRequest(result);
170                } else {
171                    LOGGER.info("Invalid request: " + new String(bytesMsg));
172                }
173            } catch (IOException ex) {
174                LOGGER.error(ex);
175            }
176        }
177
178        /**
179         * Process a discovery service request and sends back the response to the requestor.
180         * @param message the BluetoothTunnelSearchRequest
181         */
182        private void processDiscoveryRequest(BluetoothTunnelSearchRequest message) {
183            LOGGER.info("Starting search for " + message.getRequestor() + " 's request");
```

```
184
185         this.bluetoothConverter.searchServices(message.getRequest());
186
187         this.requestUUIDs.put(message.getRequest(), message.getRequestor());
188
189         ResponseObserver observer = new ResponseObserver(message);
190         observer.start();
191     }
192
193     /**
194      * Process a invoke service request and sends back the response to the requestor.
195      * @param mmessage the BluetoothTunnelSearchResult
196      */
197     private void processInvokeRequest(BluetoothTunnelSearchResult message) {
198         TunnelSearchResult result = this.bluetoothConverter.invokeActionThrowTunnel(message.getResult(
199
200         Element root = new Element("TunnelInvokeResult");
201         Document document = new Document(root);
202
203         Element requestorUUID = new Element("requestorUUID");
204         requestorUUID.setText(message.getRequestor());
205         root.addContent(requestorUUID);
206
207         Element serviceToInvoke = new Element("serviceToInvoke");
208         serviceToInvoke.setText(message.getServiceToInvoke());
209         root.addContent(serviceToInvoke);
210
211         Element actionToInvoke = new Element("actionToInvoke");
212         actionToInvoke.setText(message.getActionToInvoke());
213         root.addContent(actionToInvoke);
214
215         Element parameters = new Element("parameters");
216         root.addContent(parameters);
217
218         List<ParameterConverter> parametersConverters = result.getDeviceConverter().getServiceConverte
219         for (ParameterConverter paramConv : parametersConverters) {
220             Element parameter = new Element("parameter");
221             parameter.setAttribute("name", paramConv.getName());
222             if (paramConv.getValue() != null) {
223                 parameter.setAttribute("value", paramConv.getValue().toString());
224             }
225             parameter.setAttribute("type", paramConv.getType());
226             parameter.setAttribute("direction", paramConv.getDirection());
227
228             parameters.addContent(parameter);
229         }
230
```

```
231            XMLOutputter outputter = new XMLOutputter();
232            this.serverDevice.send(outputter.outputString(document).getBytes());
233
234            LOGGER.info("Message sent to client: " + outputter.outputString(document));
235
236        /*
237        BluetoothTunnelSearchResult response = new BluetoothTunnelSearchResult();
238        response.setRequestor(message.getRequestor());
239        response.setServiceToInvoke(message.getServiceToInvoke());
240        response.setActionToInvoke(message.getActionToInvoke());
241        response.setResult(result);
242        try {
243        this.serverDevice.send(BluetoothHandlerUtils.getBytes(response));
244        LOGGER.info("Invoke response sent");
245        } catch (IOException ex) {
246        LOGGER.error(ex);
247        }*/
248        }
249
250        /**
251         * Sends back the response of a Discovery Services Solicitation.
252         * The message will be sent in a BluetoothTunnelSearchRequest object with a TunnelSearchResponse c
253         */
254        private void sendDiscoveryResponseMessage(TunnelSearchResult result) {
255
256            this.UUIDSresult.put(this.requestUUIDs.get(result.getTunnelSearchRequest()), result);
257
258            Element root = new Element("TunnelSearchResult");
259            Document document = new Document(root);
260
261            Element requestorUUID = new Element("requestorUUID");
262            requestorUUID.setText(this.requestUUIDs.get(result.getTunnelSearchRequest()));
263            root.addContent(requestorUUID);
264
265            if (result.getDeviceConverter() != null) {
266                Element deviceConverter = new Element("deviceConverter");
267                deviceConverter.setAttribute("name", result.getDeviceConverter().getDevicename());
268                root.addContent(deviceConverter);
269
270                Set services = result.getDeviceConverter().getServices().keySet();
271                Iterator servicesIterator = services.iterator();
272
273                while (servicesIterator.hasNext()) {
274                    String serviceName = (String) servicesIterator.next();
275                    Element service = new Element("service");
276                    service.setAttribute("name", serviceName);
277                    root.addContent(service);
```

```
278
279                 Set actions = result.getDeviceConverter().getServiceConverterFromName(serviceName).get
280                 Iterator actionsIterator = actions.iterator();
281                 while (actionsIterator.hasNext()) {
282                     String actionName = (String) actionsIterator.next();
283                     Element action = new Element("action");
284                     action.setAttribute("name", actionName);
285                     service.addContent(action);
286
287                     Vector<ParameterConverter> parameters = result.getDeviceConverter().getServiceConv
288                     for (ParameterConverter param : parameters) {
289                         Element parameter = new Element("parameter");
290                         parameter.setAttribute("name", param.getName());
291                         parameter.setAttribute("direction", param.getDirection());
292                         parameter.setAttribute("type", param.getType());
293
294                         if (param.getValue() != null) {
295                             parameter.setText(param.getValue().toString());
296                         }
297                         service.addContent(parameter);
298                     }
299                 }
300             }
301         }
302
303         XMLOutputter outputter = new XMLOutputter();
304         this.serverDevice.send(outputter.outputString(document).getBytes());
305
306         LOGGER.info("Message sent to client: " + outputter.outputString(document));
307
308     /*
309     try {
310     BluetoothTunnelSearchResult response = new BluetoothTunnelSearchResult();
311     response.setRequestor(this.requestUUIDs.get(result.getTunnelSearchRequest()));
312     response.setResult(result);
313
314     this.serverDevice.send(BluetoothHandlerUtils.getBytes(response));
315     } catch (IOException ex) {
316     LOGGER.error(ex);
317     }*/
318     }
319
320     public void errorOnConnection(IOException ex) {
321         LOGGER.error(ex.getMessage(), ex);
322     }
323
324     public void errorOnReceiving(IOException ex) {
```

```
325            LOGGER.error(ex.getMessage(), ex);
326        }
327
328        public void errorOnSending(IOException ex) {
329            LOGGER.error(ex.getMessage(), ex);
330        }
331
332        private boolean isSearchRequest(byte[] bytes) throws IOException {
333            Reader reader = new StringReader(new String(bytes));
334            SAXBuilder builder = new SAXBuilder(false);
335
336            Document document;
337            try {
338                document = builder.build(reader);
339            } catch (JDOMException ex) {
340                throw new IOException(ex);
341            }
342
343            Element root = document.getRootElement();
344            if ("TunnelSearchRequest".equals(root.getName())) {
345                return true;
346            } else {
347                return false;
348            }
349        }
350
351        private boolean isInvokeRequest(byte[] bytes) throws IOException {
352            Reader reader = new StringReader(new String(bytes));
353            SAXBuilder builder = new SAXBuilder(false);
354
355            Document document;
356            try {
357                document = builder.build(reader);
358            } catch (JDOMException ex) {
359                throw new IOException(ex);
360            }
361
362            Element root = document.getRootElement();
363            if ("TunnelInvokeRequest".equals(root.getName())) {
364                return true;
365            } else {
366                return false;
367            }
368        }
369
370        /**
371         * Formato:
```

```
372     *
373     * <TunnelSearchRequest>
374     *  <requestorUUID>12345</requestorUUID>
375     *  <serviceToInvoke>Namespace</serviceToInvoke>
376     *  <actionToInvoke>Porttype</actionToInvoke>
377     * </TunnelSearchRequest>
378     *
379     * @param bytes
380     * @return
381     * @throws java.io.IOException
382     */
383    private BluetoothTunnelSearchRequest extractSearchRequest(byte[] bytes) throws IOException {
384
385        Reader reader = new StringReader(new String(bytes));
386        SAXBuilder builder = new SAXBuilder(false);
387
388        Document document;
389        try {
390            document = builder.build(reader);
391        } catch (JDOMException ex) {
392            throw new IOException(ex);
393        }
394
395        XMLOutputter outputter = new XMLOutputter();
396        LOGGER.info("Message received from client: " + outputter.outputString(document));
397
398        Element root = document.getRootElement();
399
400        Element requestorUUID = root.getChild("requestorUUID");
401        Element serviceNameSpace = root.getChild("serviceNameSpace");
402        Element servicePorttype = root.getChild("servicePorttype");
403
404        assert (requestorUUID != null);
405        assert (serviceNameSpace != null);
406        assert (servicePorttype != null);
407
408        BluetoothTunnelSearchRequest blRequest = new BluetoothTunnelSearchRequest();
409        blRequest.setRequestor(requestorUUID.getText());
410
411        TunnelSearchRequest request = new TunnelSearchRequest();
412        request.setServiceNameSpace(serviceNameSpace.getText());
413        request.setServicePorttype(servicePorttype.getText());
414        request.setTypeSearch(TunnelSearchRequest.SEARCH_SERVICE);
415        request.setDeep(TunnelSearchRequest.SEARCH_VIRTUALCACHE_AND_REMOTE);
416
417        blRequest.setRequest(request);
418
```

```
419         return blRequest;
420     }
421 
422     /**
423      * Formato:
424      *
425      * <TunnelInvokeRequest>
426      *  <requestorUUID>12345</requestorUUID>
427      *  <serviceToInvoke>Namespace</serviceToInvoke>
428      *  <actionToInvoke>Porttype</actionToInvoke>
429      *  <parameters>
430      *      <parameter name="paramName" value="paramValue"/>
431      *  </parameters>
432      * </TunnelInvokeRequest>
433      *
434      * @param bytes
435      * @return
436      * @throws java.io.IOException
437      */
438     private BluetoothTunnelSearchResult extractInvokeRequest(byte[] bytes) throws IOException {
439 
440         Reader reader = new StringReader(new String(bytes));
441         SAXBuilder builder = new SAXBuilder(false);
442 
443         Document document;
444         try {
445             document = builder.build(reader);
446         } catch (JDOMException ex) {
447             throw new IOException(ex);
448         }
449 
450         XMLOutputter outputter = new XMLOutputter();
451         LOGGER.info("Message received from client: " + outputter.outputString(document));
452 
453         Element root = document.getRootElement();
454 
455         Element requestorUUID = root.getChild("requestorUUID");
456         Element serviceToInvoke = root.getChild("serviceToInvoke");
457         Element actionToInvoke = root.getChild("actionToInvoke");
458 
459         assert (requestorUUID != null);
460         assert (serviceToInvoke != null);
461         assert (actionToInvoke != null);
462 
463         BluetoothTunnelSearchResult blResult = new BluetoothTunnelSearchResult();
464         blResult.setRequestor(requestorUUID.getText());
465         blResult.setResult(this.UUIDSresult.get(requestorUUID.getText()));
```

```
466         blResult.setServiceToInvoke(serviceToInvoke.getText());
467         blResult.setActionToInvoke(actionToInvoke.getText());
468 
469         Element parameters = root.getChild("parameters");
470         if (parameters != null) {
471             List<Element> params = parameters.getChildren();
472             for (Element param : params) {
473                 ParameterConverter paramConv = blResult.getResult().getDeviceConverter().getServiceCor
474                 paramConv.setValue(param.getAttributeValue("value"));
475             }
476         }
477 
478         return blResult;
479     }
480 
481     private class ResponseObserver extends Thread {
482 
483         BluetoothTunnelSearchRequest request;
484 
485         public ResponseObserver(BluetoothTunnelSearchRequest request) {
486             this.request = request;
487         }
488 
489         @Override
490         public void run() {
491 
492             LOGGER.info("Observer thread started for " + request.getRequestor() + " request");
493 
494             long endTime = System.currentTimeMillis() + 15000; //10 seconds
495             while (BluetoothDSBServicesProviderImpl.this.requestCache.get(request.getRequest()) == nul
496                 Thread.yield();
497             }
498 
499             TunnelSearchResult result = BluetoothDSBServicesProviderImpl.this.requestCache.get(request
500             BluetoothDSBServicesProviderImpl.this.sendDiscoveryResponseMessage(result);
501 
502             LOGGER.info("Observer thread finished");
503         }
504     }
505 
506     private class ServiceThread extends Thread {
507 
508         RFCOMMCommunicationFactory communicationFactory;
509         int maxConnections = 1000; //TODO do 1 connection at time when marge is fixed
510         boolean running = false;
511         UUID uuid = new UUID("EFA8637497899AE", false);
512 
```

```
513         @Override
514         public void run() {
515             this.communicationFactory = new RFCOMMCommunicationFactory();
516             ServerConfiguration config = new ServerConfiguration(BluetoothDSBServicesProviderImpl.this
517             config.setMaxNumberOfConnections(this.maxConnections);
518             config.setUuid(uuid);
519 
520             this.communicationFactory.waitClients(config, BluetoothDSBServicesProviderImpl.this);
521 
522             while (this.running) {
523                 Thread.yield();
524             }
525         }
526     }
527 }
```

```
01 /*
02  *
03  * $Id$
04  *
05  * The MIT License
06  *
07  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
08  *
09  * Permission is hereby granted, free of charge, to any person obtaining a copy
10  * of this software and associated documentation files (the "Software"), to deal
11  * in the Software without restriction, including without limitation the rights
12  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
13  * copies of the Software, and to permit persons to whom the Software is
14  * furnished to do so, subject to the following conditions:
15  *
16  * The above copyright notice and this permission notice shall be included in
17  * all copies or substantial portions of the Software.
18  *
19  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
20  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
21  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
22  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
23  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
24  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
25  * THE SOFTWARE.
26  */
27 package br.ufsc.inf.ppgcc.converter.bluetooth.services.messages;
28 
29 import br.ufsc.inf.ppgcc.search.TunnelSearchRequest;
```

```
30 import java.io.Serializable;
31
32 /**
33  * A entity class that encapsulates Bluetooth messages attributes.
34  * @author Fabio Kreusch <fabio at kreusch.com.br>
35  */
36 public class BluetoothTunnelSearchRequest implements Serializable {
37
38     private String requestor; //UUID of the Bluetooth service recipient
39     private TunnelSearchRequest request;
40
41     /**
42      * @return the requestor
43      */
44     public String getRequestor() {
45         return requestor;
46     }
47
48     /**
49      * @param requestor the requestor to set
50      */
51     public void setRequestor(String requestor) {
52         this.requestor = requestor;
53     }
54
55     /**
56      * @return the request
57      */
58     public TunnelSearchRequest getRequest() {
59         return request;
60     }
61
62     /**
63      * @param request the request to set
64      */
65     public void setRequest(TunnelSearchRequest request) {
66         this.request = request;
67     }
68 }
```

```
01 /*
02  *
03  * $Id$
04  *
05  * The MIT License
```

```
06  *
07  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
08  *
09  * Permission is hereby granted, free of charge, to any person obtaining a copy
10  * of this software and associated documentation files (the "Software"), to deal
11  * in the Software without restriction, including without limitation the rights
12  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
13  * copies of the Software, and to permit persons to whom the Software is
14  * furnished to do so, subject to the following conditions:
15  *
16  * The above copyright notice and this permission notice shall be included in
17  * all copies or substantial portions of the Software.
18  *
19  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
20  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
21  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
22  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
23  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
24  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
25  * THE SOFTWARE.
26  */
27 package br.ufsc.inf.ppgcc.converter.bluetooth.services.messages;
28
29 import br.ufsc.inf.ppgcc.search.TunnelSearchResult;
30 import java.io.Serializable;
31
32 /**
33  * A entity class that encapsulates Bluetooth messages attributes.
34  * @author Fabio Kreusch <fabio at kreusch.com.br>
35  */
36 public class BluetoothTunnelSearchResult  implements Serializable {
37
38     private String requestor; //UUID of the Bluetooth service recipient
39     private TunnelSearchResult result;
40     private String serviceToInvoke;
41     private String actionToInvoke;
42
43     /**
44      * @return the requestor
45      */
46     public String getRequestor() {
47         return requestor;
48     }
49
50     /**
51      * @param requestor the requestor to set
52      */
```

```
53      public void setRequestor(String requestor) {
54          this.requestor = requestor;
55      }
56
57      /**
58       * @return the result
59       */
60      public TunnelSearchResult getResult() {
61          return result;
62      }
63
64      /**
65       * @param result the result to set
66       */
67      public void setResult(TunnelSearchResult result) {
68          this.result = result;
69      }
70
71      /**
72       * @return the serviceToInvoke
73       */
74      public String getServiceToInvoke() {
75          return serviceToInvoke;
76      }
77
78      /**
79       * @param serviceToInvoke the serviceToInvoke to set
80       */
81      public void setServiceToInvoke(String serviceToInvoke) {
82          this.serviceToInvoke = serviceToInvoke;
83      }
84
85      /**
86       * @return the actionToInvoke
87       */
88      public String getActionToInvoke() {
89          return actionToInvoke;
90      }
91
92      /**
93       * @param actionToInvoke the actionToInvoke to set
94       */
95      public void setActionToInvoke(String actionToInvoke) {
96          this.actionToInvoke = actionToInvoke;
97      }
98  }
```

```
001 /*
002  *
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027
028 /*
029  * BluetoothDiscoveryServiceConsumer.java
030  *
031  * Created on 16/03/2009, 14:06:14
032  */
033 package br.ufsc.inf.bluetooth.services;
034
035 import br.ufsc.inf.ppgcc.converter.bluetooth.BluetoothHandler;
036 import br.ufsc.inf.ppgcc.converter.bluetooth.services.messages.BluetoothTunnelSearchResult;
037 import br.ufsc.inf.ppgcc.converter.entity.DeviceConverter;
038 import br.ufsc.inf.ppgcc.search.TunnelSearchResult;
039 import java.io.IOException;
040 import java.io.Reader;
041 import java.io.StringReader;
042 import java.util.Enumeration;
043 import java.util.List;
044 import javax.bluetooth.BluetoothStateException;
045 import javax.bluetooth.DeviceClass;
```

```
046 import javax.bluetooth.RemoteDevice;
047 import javax.bluetooth.ServiceRecord;
048 import javax.bluetooth.UUID;
049 import net.java.dev.marge.communication.CommunicationListener;
050 import net.java.dev.marge.entity.ClientDevice;
051 import net.java.dev.marge.entity.config.ClientConfiguration;
052 import net.java.dev.marge.factory.RFCOMMCommunicationFactory;
053 import net.java.dev.marge.inquiry.DeviceDiscoverer;
054 import net.java.dev.marge.inquiry.InquiryListener;
055 import net.java.dev.marge.inquiry.ServiceDiscoverer;
056 import net.java.dev.marge.inquiry.ServiceSearchListener;
057 import org.apache.log4j.Logger;
058 import org.jdom.Document;
059 import org.jdom.Element;
060 import org.jdom.JDOMException;
061 import org.jdom.input.SAXBuilder;
062 import org.jdom.output.XMLOutputter;
063
064 /**
065  *
066  * @author Fabio Kreusch <fabio at kreusch.com.br>
067  */
068 public class BluetoothDiscoveryServiceConsumer extends javax.swing.JFrame {
069
070     private final static Logger LOGGER = Logger.getLogger(BluetoothDiscoveryServiceConsumer.class);
071     private ServiceConnection connection;
072     private final String uuid = "EFA8637497000EA";
073
074     /** Creates new form BluetoothDiscoveryServiceConsumer */
075     public BluetoothDiscoveryServiceConsumer() {
076         initComponents();
077     }
078
079     /** This method is called from within the constructor to
080      * initialize the form.
081      * WARNING: Do NOT modify this code. The content of this method is
082      * always regenerated by the Form Editor.
083      */
084     @SuppressWarnings("unchecked")
085     // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
086     private void initComponents() {
087
088         jButton1 = new javax.swing.JButton();
089         jButton2 = new javax.swing.JButton();
090         jTextField1 = new javax.swing.JTextField();
091         jTextField2 = new javax.swing.JTextField();
092         jLabel1 = new javax.swing.JLabel();
```

```
093         jLabel2 = new javax.swing.JLabel();
094         jLabel3 = new javax.swing.JLabel();
095         jTextField3 = new javax.swing.JTextField();
096 
097         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
098 
099         jButton1.setText("Connect to service");
100         jButton1.addActionListener(new java.awt.event.ActionListener() {
101             public void actionPerformed(java.awt.event.ActionEvent evt) {
102                 jButton1ActionPerformed(evt);
103             }
104         });
105 
106         jButton2.setText("Disconnect from service");
107         jButton2.addActionListener(new java.awt.event.ActionListener() {
108             public void actionPerformed(java.awt.event.ActionEvent evt) {
109                 jButton2ActionPerformed(evt);
110             }
111         });
112 
113         jTextField1.setText("FakeConverters");
114         jTextField1.addActionListener(new java.awt.event.ActionListener() {
115             public void actionPerformed(java.awt.event.ActionEvent evt) {
116                 jTextField1ActionPerformed(evt);
117             }
118         });
119 
120         jTextField2.setText("FAKE_SERVICE");
121         jTextField2.addActionListener(new java.awt.event.ActionListener() {
122             public void actionPerformed(java.awt.event.ActionEvent evt) {
123                 jTextField2ActionPerformed(evt);
124             }
125         });
126 
127         jLabel1.setText("Namespace");
128 
129         jLabel2.setText("Porttype");
130 
131         jLabel3.setText("Msg. Size (kb)");
132 
133         jTextField3.setText("10000");
134         jTextField3.addActionListener(new java.awt.event.ActionListener() {
135             public void actionPerformed(java.awt.event.ActionEvent evt) {
136                 jTextField3ActionPerformed(evt);
137             }
138         });
139 
```

```
140        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
141        getContentPane().setLayout(layout);
142        layout.setHorizontalGroup(
143            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
144            .addGroup(layout.createSequentialGroup()
145                .addContainerGap()
146                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
147                    .addComponent(jLabel1)
148                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
149                        .addComponent(jLabel3)
150                        .addComponent(jLabel2)))
151                .addGap(16, 16, 16)
152                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
153                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, fa
154                        .addComponent(jButton1)
155                        .addComponent(jButton2)
156                        .addComponent(jTextField1, javax.swing.GroupLayout.DEFAULT_SIZE, 178, Short.MA
157                        .addComponent(jTextField2))
158                    .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.Gro
159                .addContainerGap(115, Short.MAX_VALUE))
160        );
161        layout.setVerticalGroup(
162            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
163            .addGroup(layout.createSequentialGroup()
164                .addGap(17, 17, 17)
165                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
166                    .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.Gro
167                    .addComponent(jLabel1))
168                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
169                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
170                    .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.Gro
171                    .addComponent(jLabel2))
172                .addGap(18, 18, 18)
173                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
174                    .addComponent(jLabel3)
175                    .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.Gro
176                .addGap(5, 5, 5)
177                .addComponent(jButton1)
178                .addGap(18, 18, 18)
179                .addComponent(jButton2)
180                .addContainerGap(111, Short.MAX_VALUE))
181        );
182
183        pack();
184    }// </editor-fold>//GEN-END:initComponents
185
186    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton2Ac
```

```
187         this.connection.disconnect();
188     }//GEN-LAST:event_jButton2ActionPerformed
189 
190     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton1Ac
191         if (this.connection == null) {
192             this.connection = new ServiceConnection();
193         }
194         this.connection.sendMessage();
195     }//GEN-LAST:event_jButton1ActionPerformed
196 
197     private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jTextFi
198         // TODO add your handling code here:
199     }//GEN-LAST:event_jTextField1ActionPerformed
200 
201     private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jTextFi
202         // TODO add your handling code here:
203     }//GEN-LAST:event_jTextField2ActionPerformed
204 
205     private void jTextField3ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jTextFi
206         // TODO add your handling code here:
207     }//GEN-LAST:event_jTextField3ActionPerformed
208 
209     /**
210      * @param args the command line arguments
211      */
212     public static void main(String args[]) {
213         java.awt.EventQueue.invokeLater(new Runnable() {
214 
215             public void run() {
216                 new BluetoothDiscoveryServiceConsumer().setVisible(true);
217             }
218         });
219     }
220 
221     // Variables declaration - do not modify//GEN-BEGIN:variables
222     private javax.swing.JButton jButton1;
223     private javax.swing.JButton jButton2;
224     private javax.swing.JLabel jLabel1;
225     private javax.swing.JLabel jLabel2;
226     private javax.swing.JLabel jLabel3;
227     private javax.swing.JTextField jTextField1;
228     private javax.swing.JTextField jTextField2;
229     private javax.swing.JTextField jTextField3;
230     // End of variables declaration//GEN-END:variables
231 
232     private class ServiceConnection implements InquiryListener, ServiceSearchListener, CommunicationLi
233 
```

```
234         ClientDevice clientDevice;
235         String message = "Hellloooo!!!";
236         private RemoteDevice remoteDevice = null;
237         private ServiceRecord[] services = null;
238
239         public void sendMessage() {
240             try {
241                 int size = new Integer(BluetoothDiscoveryServiceConsumer.this.jTextField3.getText());
242
243                 StringBuffer msgBuffer = new StringBuffer();
244                 for (int i = 0; i < size; i++) {
245                     msgBuffer.append("b");
246                 }
247
248                 this.message = msgBuffer.toString();
249                 LOGGER.info("Message size: " + this.message.length());
250
251                 if (this.remoteDevice == null) {
252                     LOGGER.info("Searching for devices...");
253                     DeviceDiscoverer.getInstance().startInquiryGIAC(this);
254                 } else {
255                     this.serviceSearchCompleted(remoteDevice, services);
256                 }
257
258             } catch (BluetoothStateException ex) {
259                 LOGGER.error(ex.getMessage(), ex);
260             }
261         }
262
263         public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass) {
264             LOGGER.info("Device discovered: " + remoteDevice.getBluetoothAddress());
265         }
266
267         public void inquiryCompleted(RemoteDevice[] remoteDevices) {
268             try {
269                 UUID[] uuids = {new UUID("EFA8637497899AE", false)};
270                 ServiceDiscoverer.getInstance().startSearch(uuids, remoteDevices[0], this);
271                 LOGGER.info("Starting service search");
272             } catch (BluetoothStateException ex) {
273                 LOGGER.error(ex.getMessage(), ex);
274             }
275         }
276
277         public void inquiryError() {
278             LOGGER.error("Error during Inquiry process");
279         }
280
```

```
281        public void serviceSearchCompleted(RemoteDevice remoteDevice, ServiceRecord[] services) {
282            try {
283                LOGGER.info("Service search completed");
284
285                this.remoteDevice = remoteDevice;
286                this.services = services;
287
288                RFCOMMCommunicationFactory communicationFactory = new RFCOMMCommunicationFactory();
289                ClientConfiguration clientConfiguration = new ClientConfiguration(services[0], this);
290                this.clientDevice = communicationFactory.connectToServer(clientConfiguration);
291                this.clientDevice.startListening();
292
293                this.requestServiceSearch();
294
295                LOGGER.info("Message sent to server");
296            } catch (IOException ex) {
297                LOGGER.error(ex.getMessage(), ex);
298            }
299        }
300
301        private void requestServiceSearch() {
302            //try {
303
304            Element root = new Element("TunnelSearchRequest");
305            Document document = new Document(root);
306
307            Element requestorUUID = new Element("requestorUUID");
308            requestorUUID.setText(BluetoothDiscoveryServiceConsumer.this.uuid);
309            root.addContent(requestorUUID);
310
311            Element serviceNameSpace = new Element("serviceNameSpace");
312            serviceNameSpace.setText(BluetoothDiscoveryServiceConsumer.this.jTextField1.getText());
313            root.addContent(serviceNameSpace);
314
315            Element servicePorttype = new Element("servicePorttype");
316            servicePorttype.setText(BluetoothDiscoveryServiceConsumer.this.jTextField2.getText());
317            root.addContent(servicePorttype);
318
319            XMLOutputter outputter = new XMLOutputter();
320
321            LOGGER.info("Requesting service search");
322            this.clientDevice.send(outputter.outputString(document).getBytes());
323        }
324
325        public void deviceNotReachable() {
326            LOGGER.error("Device not reachable");
327        }
```

```
328
329         public void serviceSearchError() {
330             LOGGER.error("Service search error");
331         }
332
333         public void receiveMessage(byte[] bytesMessage) {
334
335             //LOGGER.info("Message received from server: " + new String(bytesMessage));
336
337             Reader reader = new StringReader(new String(bytesMessage));
338             SAXBuilder builder = new SAXBuilder(false);
339
340             try {
341                 Document response = builder.build(reader);
342                 Element responseRoot = response.getRootElement();
343
344                 if ("TunnelSearchResult".equals(responseRoot.getName())) {
345                     LOGGER.info("Service found");
346
347                     Element root = new Element("TunnelInvokeRequest");
348                     Document document = new Document(root);
349
350                     Element requestorUUID = new Element("requestorUUID");
351                     requestorUUID.setText(BluetoothDiscoveryServiceConsumer.this.uuid);
352                     root.addContent(requestorUUID);
353
354                     String service = BluetoothDiscoveryServiceConsumer.this.jTextField2.getText();
355
356                     Element serviceToInvoke = new Element("serviceToInvoke");
357                     serviceToInvoke.setText(service);
358                     root.addContent(serviceToInvoke);
359
360                     Element actionToInvoke = new Element("actionToInvoke");
361                     actionToInvoke.setText(service);
362                     root.addContent(actionToInvoke);
363
364                     Element parameters = new Element("parameters");
365                     parameters.setText(service);
366                     root.addContent(parameters);
367
368                     Element inputParam = new Element("parameter");
369                     inputParam.setAttribute("name", BluetoothHandler.BLUETOOTH_PARAMETER_NAME_INPUT);
370                     inputParam.setAttribute("value", message);
371                     parameters.addContent(inputParam);
372
373                     XMLOutputter outputter = new XMLOutputter();
374
```

```
375                     LOGGER.info("Requesting service invocation");
376                     this.clientDevice.send(outputter.outputString(document).getBytes());
377                 } else if ("TunnelInvokeResult".equals(responseRoot.getName())) {
378                     LOGGER.info("Service invocation result received");
379 
380                     Element parameters = responseRoot.getChild("parameters");
381                     List<Element> children = parameters.getChildren();
382 
383                     for (Element child : children) {
384                         //LOGGER.info("Parameter " + child.getAttributeValue("name") + " value: " + ch
385                     }
386 
387                     this.clientDevice.stopListenning();
388                     this.clientDevice.close();
389                 } else {
390                     LOGGER.error("Invalid response");
391                 }
392             } catch (IOException ex) {
393                 LOGGER.error(ex);
394             } catch (JDOMException ex) {
395                 LOGGER.error(ex);
396             }
397         }
398 
399         public void errorOnReceiving(IOException ex) {
400             LOGGER.error(ex.getMessage(), ex);
401         }
402 
403         public void errorOnSending(IOException ex) {
404             LOGGER.error(ex.getMessage(), ex);
405         }
406 
407         public void disconnect() {
408             this.clientDevice.stopListenning();
409             this.clientDevice.close();
410 
411             LOGGER.info("Disconnected from server");
412         }
413 
414         private String extractResponse(BluetoothTunnelSearchResult message) {
415 
416             TunnelSearchResult result = message.getResult();
417             String response = "UUID: " + message.getRequestor();
418 
419             DeviceConverter deviceConverter = result.getDeviceConverter();
420 
421             if (deviceConverter != null) {
```

```
422                     response += ", Device: " + deviceConverter.getDevicename() + ", Services: [";
423                     if (deviceConverter != null) {
424                         Enumeration serviceKeys = deviceConverter.getServices().keys();
425                         while (serviceKeys.hasMoreElements()) {
426                             response += " " + (String) serviceKeys.nextElement();
427                         }
428                     }
429                 }
430 
431                 response += "]";
432 
433                 return response;
434             }
435         }
436 }
```

```
001 /*
002  *
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027 package br.ufsc.inf.bluetooth.services;
028 
029 import java.io.IOException;
```

```
030 import java.util.ArrayList;
031 import java.util.List;
032 import javax.bluetooth.RemoteDevice;
033 import javax.bluetooth.UUID;
034 import net.java.dev.marge.communication.CommunicationListener;
035 import net.java.dev.marge.communication.ConnectionListener;
036 import net.java.dev.marge.entity.ServerDevice;
037 import net.java.dev.marge.entity.config.ServerConfiguration;
038 import net.java.dev.marge.factory.CommunicationFactory;
039 import net.java.dev.marge.factory.RFCOMMCommunicationFactory;
040 import org.apache.log4j.Logger;
041
042 /**
043  * A Bluetooth service that echoes the received message.
044  * An instance of EchoBluetoothService is retrieved with a call to getInstance().
045  * @author Fabio Kreusch <fabio at kreusch.com.br>
046  */
047 public class EchoBluetoothService implements CommunicationListener, ConnectionListener {
048
049     private final static Logger LOGGER = Logger.getLogger(EchoBluetoothService.class);
050     public static final String SERVICE_NAME = "EchoService";
051     public static final UUID SERVICE_UUID = new UUID("EFA8637495932EF", false);
052     private static EchoBluetoothService bluetoothService;
053     private ServerDevice serverDevice;
054     private List<RemoteDevice> clientDevices = new ArrayList<RemoteDevice>();
055     private ServiceThread serviceThread;
056
057     private EchoBluetoothService() {
058     }
059
060     /**
061      * Returns an instance of EchoBluetoothService.
062      * @return EchoBluetoothService instance.
063      */
064     public static EchoBluetoothService getInstance() {
065         if (EchoBluetoothService.bluetoothService == null) {
066             EchoBluetoothService.bluetoothService = new EchoBluetoothService();
067         }
068         return EchoBluetoothService.bluetoothService;
069     }
070
071     /**
072      * Starts this service and waits for connections.
073      * @param maxConnections The maximum number of connections that this service will accept.
074      */
075     public void startService(int maxConnections) {
076         if (this.serviceThread == null) {
```

```
077             this.serviceThread = new ServiceThread();
078             this.serviceThread.maxConnections = maxConnections;
079             this.serviceThread.start();
080             LOGGER.info("BluetoothService '" + SERVICE_NAME + " : " + SERVICE_UUID + "' is now waiting
081         } else {
082             LOGGER.info("BluetoothService '" + SERVICE_NAME + " : " + SERVICE_UUID + "' already starte
083         }
084     }
085 
086     /**
087      * Starts this service and waits for connections.
088      * This method will make the service to accept only one connection.
089      */
090     public void startService() {
091         this.startService(1);
092     }
093 
094     /**
095      * Stops a service.
096      */
097     public void stopService() {
098         if (this.getServerDevice() != null) {
099             this.getServerDevice().stopListenning();
100             this.getServerDevice().close();
101             this.serverDevice = null;
102             this.getClientDevices().clear();
103             LOGGER.info("Stoped server device");
104         }
105 
106         if ((this.serviceThread != null)) {
107             this.serviceThread.running = false;
108             this.serviceThread.interrupt();
109 
110             try {
111                 this.serviceThread.join();
112             } catch (InterruptedException ex) {
113                 LOGGER.debug(ex);
114             }
115 
116             this.serviceThread = null;
117             LOGGER.info("Stoped BluetoothService thread");
118         } else {
119             LOGGER.info("No active BluetoothService");
120         }
121     }
122 
123     /**
```

```
124      * Listener method for a new connection.
125      * @param server The ServerDevice object
126      * @param client The RemoteDevice object
127      */
128     public void connectionEstablished(ServerDevice server, RemoteDevice client) {
129 
130         if (this.getServerDevice() == null) {
131             this.serverDevice = server;
132             this.getServerDevice().startListening();
133             this.getServerDevice().setEnableBroadcast(true);
134         }
135 
136         this.getClientDevices().add(client);
137 
138         LOGGER.info("Connection received from " + client.getBluetoothAddress());
139     }
140 
141     public void errorOnConnection(IOException ex) {
142         LOGGER.error("Error on connection: " + ex.getMessage(), ex);
143     }
144 
145     public void receiveMessage(byte[] msgBytes) {
146         this.getServerDevice().send(this.processReturnMessage(msgBytes));
147     }
148 
149     public void errorOnReceiving(IOException ex) {
150         LOGGER.error("Error on receiving: " + ex.getMessage(), ex);
151     }
152 
153     public void errorOnSending(IOException ex) {
154         LOGGER.error("Error on sending: " + ex.getMessage(), ex);
155     }
156 
157     private byte[] processReturnMessage(byte[] msgBytes) {
158         String msg = new String(msgBytes);
159         LOGGER.info("Received message: " + msg);
160         msg = "Echoed: " + msg;
161         LOGGER.info("Returned message: " + msg);
162         return msg.getBytes();
163     }
164 
165     /**
166      * @return the server
167      */
168     public ServerDevice getServerDevice() {
169         return serverDevice;
170     }
```

```
171
172     /**
173      * @return the clients
174      */
175     public List<RemoteDevice> getClientDevices() {
176         return clientDevices;
177     }
178
179     public void initialisationSucessful() {
180         LOGGER.info("Service initialized");
181     }
182
183     private class ServiceThread extends Thread {
184
185         CommunicationFactory communicationFactory;
186         int maxConnections = 1;
187         boolean running = false;
188
189         @Override
190         public void run() {
191             try {
192                 this.communicationFactory = new RFCOMMCommunicationFactory();
193                 ServerConfiguration config = new ServerConfiguration(EchoBluetoothService.this, EchoBl
194                 config.setMaxNumberOfConnections(this.maxConnections);
195                 config.setServerName(SERVICE_NAME);
196
197                 this.communicationFactory.waitClients(config, EchoBluetoothService.this);
198
199                 while (this.running) {
200                     Thread.yield();
201                 }
202             } catch (Exception ex) {
203                 LOGGER.fatal(ex.getMessage(), ex);
204             }
205         }
206     }
207
208     public static void main(String[] args) {
209         EchoBluetoothService.getInstance().startService(7);
210     }
211 }
```

```
    001 /*
002  *
003  * $Id$
```

```
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027
028 /*
029  * EchoBluetoothServiceConsumer.java
030  *
031  * Created on 02/03/2009, 08:07:48
032  */
033 package br.ufsc.inf.bluetooth.services;
034
035 import java.io.IOException;
036 import javax.bluetooth.BluetoothStateException;
037 import javax.bluetooth.DeviceClass;
038 import javax.bluetooth.RemoteDevice;
039 import javax.bluetooth.ServiceRecord;
040 import javax.bluetooth.UUID;
041 import net.java.dev.marge.communication.CommunicationListener;
042 import net.java.dev.marge.entity.ClientDevice;
043 import net.java.dev.marge.entity.config.ClientConfiguration;
044 import net.java.dev.marge.factory.RFCOMMCommunicationFactory;
045 import net.java.dev.marge.inquiry.DeviceDiscoverer;
046 import net.java.dev.marge.inquiry.InquiryListener;
047 import net.java.dev.marge.inquiry.ServiceDiscoverer;
048 import net.java.dev.marge.inquiry.ServiceSearchListener;
049 import org.apache.log4j.Logger;
050
```

```
051 /**
052  *
053  * @author Fabio Kreusch <fabio at kreusch.com.br>
054  */
055 public class EchoBluetoothServiceConsumer extends javax.swing.JDialog {
056
057     private final static Logger LOGGER = Logger.getLogger(EchoBluetoothServiceConsumer.class);
058     private ServiceConnection connection;
059
060     /** Creates new form EchoBluetoothServiceConsumer */
061     public EchoBluetoothServiceConsumer(java.awt.Frame parent, boolean modal) {
062         super(parent, modal);
063         initComponents();
064     }
065
066     /** This method is called from within the constructor to
067      * initialize the form.
068      * WARNING: Do NOT modify this code. The content of this method is
069      * always regenerated by the Form Editor.
070      */
071     @SuppressWarnings("unchecked")
072     // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
073     private void initComponents() {
074
075         jButton1 = new javax.swing.JButton();
076         jButton2 = new javax.swing.JButton();
077         jButton3 = new javax.swing.JButton();
078
079         setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
080
081         jButton1.setText("Conect to server");
082         jButton1.addActionListener(new java.awt.event.ActionListener() {
083             public void actionPerformed(java.awt.event.ActionEvent evt) {
084                 jButton1ActionPerformed(evt);
085             }
086         });
087
088         jButton2.setText("Disconnect from server");
089         jButton2.addActionListener(new java.awt.event.ActionListener() {
090             public void actionPerformed(java.awt.event.ActionEvent evt) {
091                 jButton2ActionPerformed(evt);
092             }
093         });
094
095         jButton3.setText("Send hello");
096         jButton3.addActionListener(new java.awt.event.ActionListener() {
097             public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
098                 jButton3ActionPerformed(evt);
099             }
100         });
101 
102         javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
103         getContentPane().setLayout(layout);
104         layout.setHorizontalGroup(
105             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
106             .addGroup(layout.createSequentialGroup()
107                 .addGap(32, 32, 32)
108                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
109                     .addComponent(jButton3)
110                     .addComponent(jButton2)
111                     .addComponent(jButton1))
112                 .addContainerGap(92, Short.MAX_VALUE))
113         );
114         layout.setVerticalGroup(
115             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
116             .addGroup(layout.createSequentialGroup()
117                 .addGap(45, 45, 45)
118                 .addComponent(jButton1)
119                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
120                 .addComponent(jButton3)
121                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.Group
122                 .addComponent(jButton2)
123                 .addContainerGap())
124         );
125 
126         pack();
127     }// </editor-fold>//GEN-END:initComponents
128 
129     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton1Ac
130         connection = new ServiceConnection();
131     }//GEN-LAST:event_jButton1ActionPerformed
132 
133     private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton2Ac
134         connection.disconnect();
135     }//GEN-LAST:event_jButton2ActionPerformed
136 
137     private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton3Ac
138         connection.sendMessage();
139     }//GEN-LAST:event_jButton3ActionPerformed
140 
141     /**
142      * @param args the command line arguments
143      */
144     public static void main(String args[]) {
```

```
145         java.awt.EventQueue.invokeLater(new Runnable() {
146 
147             public void run() {
148                 EchoBluetoothServiceConsumer dialog = new EchoBluetoothServiceConsumer(new javax.swing
149                 dialog.addWindowListener(new java.awt.event.WindowAdapter() {
150 
151                     @Override
152                     public void windowClosing(java.awt.event.WindowEvent e) {
153                         System.exit(0);
154                     }
155                 });
156                 dialog.setVisible(true);
157             }
158         });
159     }
160 
161     // Variables declaration - do not modify//GEN-BEGIN:variables
162     private javax.swing.JButton jButton1;
163     private javax.swing.JButton jButton2;
164     private javax.swing.JButton jButton3;
165     // End of variables declaration//GEN-END:variables
166 
167     private class ServiceConnection implements InquiryListener, ServiceSearchListener, CommunicationLi
168 
169         ClientDevice clientDevice;
170         ClientDevice clientDevice2;
171 
172         public ServiceConnection() {
173             try {
174                 DeviceDiscoverer.getInstance().startInquiryGIAC(this);
175             } catch (BluetoothStateException ex) {
176                 LOGGER.error(ex.getMessage(), ex);
177             }
178         }
179 
180         public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass) {
181             LOGGER.info("Device discovered: " + remoteDevice.getBluetoothAddress());
182         }
183 
184         public void inquiryCompleted(RemoteDevice[] remoteDevices) {
185             try {
186                 UUID[] uuids = {new UUID("EFA8637495932EF", false)};
187 
188                 for (RemoteDevice remoteDevice : remoteDevices) {
189                     ServiceDiscoverer.getInstance().startSearch(uuids, remoteDevice, this);
190                 }
191 
```

```
192                 LOGGER.info("Starting service search");
193             } catch (BluetoothStateException ex) {
194                 LOGGER.error(ex.getMessage(), ex);
195             }
196         }
197 
198         public void inquiryError() {
199             LOGGER.error("Error during Inquiry process");
200         }
201 
202         public synchronized void serviceSearchCompleted(RemoteDevice remoteDevice, ServiceRecord[] se
203             try {
204                 LOGGER.info("Service search completed");
205 
206                 RFCOMMCommunicationFactory communicationFactory = new RFCOMMCommunicationFactory();
207                 ClientConfiguration clientConfiguration = new ClientConfiguration(services[0], this);
208 
209                 if (this.clientDevice == null) {
210                     this.clientDevice = communicationFactory.connectToServer(clientConfiguration);
211                     this.clientDevice.startListening();
212                     LOGGER.info("Connected to service 1");
213                 //this.clientDevice.send("Hello World".getBytes());
214                 } else {
215                     this.clientDevice2 = communicationFactory.connectToServer(clientConfiguration);
216                     this.clientDevice2.startListening();
217                     LOGGER.info("Connected to service 2");
218                 //this.clientDevice2.send("Hello World".getBytes());
219                 }
220 
221                 //LOGGER.info("Message sent to server");
222             } catch (IOException ex) {
223                 LOGGER.error(ex.getMessage(), ex);
224             }
225         }
226 
227         public void sendMessage() {
228             this.clientDevice.send("Hello from client1".getBytes());
229             this.clientDevice2.send("Hello from client2".getBytes());
230         }
231 
232         public void deviceNotReachable() {
233             LOGGER.error("Device not reachable");
234         }
235 
236         public void serviceSearchError() {
237             LOGGER.error("Service search error");
238         }
```

```
239
240         public void receiveMessage(byte[] msg) {
241             LOGGER.info("Message received: " + new String(msg));
242         }
243
244         public void errorOnReceiving(IOException ex) {
245             LOGGER.error(ex.getMessage(), ex);
246         }
247
248         public void errorOnSending(IOException ex) {
249             LOGGER.error(ex.getMessage(), ex);
250         }
251
252         public void disconnect() {
253             this.clientDevice.stopListenning();
254             this.clientDevice.close();
255
256             LOGGER.info("Disconnected from server");
257         }
258     }
259 }
```

```
01 /*
02  *
03  * $Id$
04  *
05  * The MIT License
06  *
07  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
08  *
09  * Permission is hereby granted, free of charge, to any person obtaining a copy
10  * of this software and associated documentation files (the "Software"), to deal
11  * in the Software without restriction, including without limitation the rights
12  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
13  * copies of the Software, and to permit persons to whom the Software is
14  * furnished to do so, subject to the following conditions:
15  *
16  * The above copyright notice and this permission notice shall be included in
17  * all copies or substantial portions of the Software.
18  *
19  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
20  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
21  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
22  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
23  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
24  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
25  * THE SOFTWARE.
26  */
27
28 package br.ufsc.inf.bluetooth.services;
29
30 /**
31  *
32  * @author Fabio Kreusch <fabio at kreusch.com.br>
33  */
34 public class Initializer {
35     public static void main(String args[]) throws Exception {
36         StackEmulator.getInstance().startServer();
37         EchoBluetoothService.getInstance().startService(7); //TODO do 1 connection at time when marge i
38     }
39 }
```

```
001 /*
002  *
003  * $Id$
004  *
005  * The MIT License
006  *
007  * Copyright (c) 2009 Fabio Kreusch <fabio at kreusch.com.br>
008  *
009  * Permission is hereby granted, free of charge, to any person obtaining a copy
010  * of this software and associated documentation files (the "Software"), to deal
011  * in the Software without restriction, including without limitation the rights
012  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
013  * copies of the Software, and to permit persons to whom the Software is
014  * furnished to do so, subject to the following conditions:
015  *
016  * The above copyright notice and this permission notice shall be included in
017  * all copies or substantial portions of the Software.
018  *
019  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
020  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
021  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
022  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
023  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
024  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
025  * THE SOFTWARE.
026  */
027 package br.ufsc.inf.bluetooth.services;
028
```

```
029 import org.apache.log4j.Logger;
030 
031 /**
032  * Provides access to the BlueCove stack emulator.
033  * An instance of a StackEmulator is retrieved with a call to getInstance().
034  * @author Fabio Kreusch <fabio at kreusch.com.br>
035  */
036 public class StackEmulator {
037 
038     private final static Logger LOGGER = Logger.getLogger(StackEmulator.class);
039     private static StackEmulator emulator;
040     private EmulatorThread thread;
041 
042     private StackEmulator() {
043     }
044 
045     /**
046      * Gets an instance of StackEmulator.
047      * @return StackEmulator instance.
048      */
049     public static StackEmulator getInstance() {
050         if (StackEmulator.emulator == null) {
051             StackEmulator.emulator = new StackEmulator();
052         }
053         return StackEmulator.emulator;
054     }
055 
056     /**
057      * Starts the BlueCove stack emulator.
058      */
059     public void startServer() throws Exception {
060         if (StackEmulator.getInstance().thread == null) {
061             StackEmulator.getInstance().thread = new EmulatorThread();
062             StackEmulator.getInstance().thread.running = true;
063             StackEmulator.getInstance().thread.start();
064             LOGGER.info("StackEmulator started");
065         } else {
066             LOGGER.info("StackEmulator already started");
067         }
068     }
069 
070     /**
071      * Stops the BlueCove stack emulator.
072      */
073     public void stopServer() {
074         if ((StackEmulator.getInstance().thread != null) && (StackEmulator.getInstance().thread.isAli
075             StackEmulator.getInstance().thread.running = false;
```

```
076             StackEmulator.getInstance().thread.interrupt();
077             try {
078                 StackEmulator.getInstance().thread.join();
079             } catch (InterruptedException ex) {
080                 LOGGER.debug(ex);
081             }
082 
083             LOGGER.info("StackEmulator stoped");
084         } else {
085             LOGGER.info("No active StackEmulator");
086         }
087 
088         StackEmulator.getInstance().thread = null;
089     }
090 
091     private class EmulatorThread extends Thread {
092 
093         private boolean running = false;
094 
095         @Override
096         public void run() {
097             try {
098                 com.intel.bluetooth.emu.EmuServer.main(new String[0]);
099                 while (this.running) {
100                     Thread.sleep(1000);
101                 }
102             } catch (InterruptedException ex) {
103                 LOGGER.debug(ex.getMessage(), ex);
104             } catch (Exception ex) {
105                 LOGGER.fatal(ex.getMessage(), ex);
106             }
107         }
108     }
109 
110     public static void main(String[] args) throws Exception {
111         StackEmulator.getInstance().startServer();
112     }
113 }
```