UNIVERSIDADE FEDERAL DE SANTA CATARINA

SISTEMAS DE INFORMAÇÃO

INE5632 – Projetos II

Uma Implementação do Device Service Bus (DSB) em um ambiente .NET

Autor

Victor de Oliveira Bernardino

Orientador

Prof. Frank Augusto Siqueira

Florianópolis 2009

Lista de Acrônimos

XML - Xtensible Markup Language

DSB – Device Service Bus

DPWS – Devices Profile for Web Services

Middleware - Camada de Software que abstrai ações complexas

Framework - Conjunto de suporte a construção de software

.NET - Um framework da Microsoft

C# - Uma das linguagens de programação utilizadas pelo .NET

JVM - Java Virtual Machine

SOA - Service-oriented architecture

SOAP - Simple Object Access Protocol

WSDL- Web Service Definition Language

API - Application Programming Interface

SGML - Standard Generalized Markup Language

HTML - HyperText Markup Language

XHTML - eXtensible Hypertext Markup Language

CSS - Cascading Style Sheets

UTF-8 - 8bit Unicode Transformation Format

OASIS - Organization for the Advancement of Structured Information Standards

CLR - Common Language Runtime

F#.NET - Uma linguagem de programação nativa do framework .NET

SmalTalk.NET - Um suporte a linguagem Smalltalk do framework .NET

J2ME - Java Plataform Micro Edition

CTC - Centro Tecnológico da Universidade Federal de Santa Catarina

RFID - Radio-Frequency IDentification

Sumário

1. Introdução

Num futuro próximo poderemos ter dispositivos móveis, celulares, PDA´s. conectados a diversos provedores de serviços simultaneamente, sendo que os próprios dispositivos móveis podem tornar-se provedores de serviços.

Com esta frase pode-se ter uma breve visão sobre o que será apresentado neste trabalho de conclusão de curso, baseado no trabalho de mestrado do Gustavo Medeiros Araújo, orientado pelo professor Frank Siqueira, que vem com intuito de ampliar a plataforma de suporte do *middleware* DSB(*Device Service Bus*).

O DSB permite a interação entre aparelhos com limitação de recursos computacionais e que adotam tipos de rede diferentes entre si, possibilitando a comunicação entre aparelhos heterogêneos, de uma maneira simples, transparente e sem garantias de que a conexão estará sempre ativa.

De uma forma transparente similar ao *plug-and-play*, o DSB adiciona um novo membro descoberto, verifica os serviços oferecidos por ele e facilita a comunicação.

1.1.Motivação

Com o passar dos anos os computadores foram submetidos a uma forte tendência de miniaturização. Seguindo essa tendência, alguns começaram a funcionar com uma fonte de energia acoplada para que pudessem ser deslocados facilmente de forma que pudessem ser carregados.

Hoje existem milhões de aparelhos computacionais embarcados, alguns com muito poder de processamento, outros com pouco poder de processamento.

Neste trabalho iremos abordar uma implementação de *middleware* que proporciona uma infra-estrutura de software, que possibilita um melhor uso dos recursos de hardware de forma distribuída, fazendo com que dispositivos com grandes restrições de hardware possam realizar tarefas que talvez não fossem possíveis ou que não fossem viáveis para determinado dispositivo. Através de troca de mensagens, via rede ou mesmo via internet, essa tecnologia permite a descoberta e provêem meios para acesso aos serviços disponibilizados na rede.

A motivação principal se dá pelo fato de dispositivos computacionais com grandes limitações poderem utilizar serviços de outros, e com isso enriquecer suas utilidades aos usuários. Tal fator torna esta área muito mais atrativa para seus usuários e empresas, pois muito processamento de dados poderá ser evitado nestes dispositivos pelo processamento alheio das informações requeridas.

1.2.Objetivos

Nesta sessão são apresentados os objetivos deste trabalho, objetivos gerais e objetivos específicos respectivamente.

1.2.1. Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma infra-estrutura de *middleware* compatível ao DSB utilizando a tecnologia .NET, e que possibilite a comunicação entre dispositivos computacionais embarcados que utilizem diferentes tipos de comunicação.

1.2.2. Objetivos específicos

A importância de incrementar a gama de plataformas nas quais um software pode ser utilizado torna-se muito grande quando visamos atingir um grande número de usuários. Seguindo este princípio, este projeto vem com intuito de aumentar a diversidade de dispositivos que podem utilizar o *middleware* DSB.

Em uma rede sem garantias de entrega de mensagens, no caso do DSB, é vital garantir que dispositivos computacionais com serviços que estejam ao alcance fiquem atualizados na lista de serviços disponíveis.

Em um sistema ubíquo sabe-se que novos dispositivos podem ser encontrados a cada instante sem que haja uma garantia de permanência na rede, e justamente por não saber a intenção de cada novo participante é muito importante sempre manter um bom padrão de segurança nas informações transmitidas, impedindo acesso a informações sigilosas na rede via DSB.

Sabe-se que quanto mais complexos os sistemas desenvolvidos, mais recursos devem ser alocados para sua execução. Por isso, tem-se o objetivo de tentar tornar o mais simples possível a interação das mensagens entre os provedores de serviço e os consumidores de serviço.

1.3.Justificativa

A proposta inicial é implementar parte das funções do *middleware* DSB usando a tecnologia .NET empregando a linguagem C#. Nem todas as funcionalidades serão implementadas; por exemplo, o suporte a conexão via Bluetooth não será portado.

Este trabalho se justifica por tratar de um tema muito atual e em desenvolvimento. O DSB tem como ponto forte sua utilização por dispositivos móveis com limitações computacionais, porém estes dispositivos devem ter alguma versão da JVM instalado, mesmo podendo utilizar serviços de aparelhos que não utilizam o DSB este *middleware* ficou limitado a um número mais reduzido de dispositivos.

Com o desenvolvimento a ser feito sobre a plataforma .NET, a gama de aparelhos nos quais poderá ser utilizado o DSB aumentará, e assim o mesmo poderá ser melhor difundido e utilizado.

1.4. Metodologia

A princípio será estudado o DSB e seus cinco componentes: *Converter*, *Bridge*, *Tunnel Search Manager*, *Virtual Device Cache* e *Device Tunnel* , assim como a especificação DPWS, que se tornou um padrão e por conseqüência estaremos utilizando dentro do trabalho.

Será também estudado como utilizar o DPWS dentro do framework .NET versão 3.5, quais as limitações e benefícios de se utilizá-lo.

Como o DPWS é independente de plataforma e de linguagem de programação, faremos a migração de plataformas do DSB, que foi desenvolvido em JAVA, para usar .NET, visando assim atender uma maior gama de aparelhos móveis, aumentando consideravelmente a inclusão de novos dispositivos móveis suportados.

Também serão estudados e expostos alguns dos padrões utilizados no DPWS para um melhor entendimento da sua utilização e abrangência.

1.5.Resultados Esperados

As expectativas em torno deste trabalho giram em torno da compatibilidade e comunicação entre as diferentes implementações do *middleware* desenvolvidas, ou seja, entre a primeira implementação do DSB, que a princípio foi desenvolvida usando a linguagem JAVA, e a nova implementação do DSB desenvolvido neste trabalho, que utiliza a plataforma .NET.

1.6.Estrutura do Documento

Nas próximas seções veremos um pouco sobre cada tecnologia abordada no *middleware* DSB.

No capítulo 2 serão abordadas algumas das tecnologias padronizadas que possibilitaram o desenvolvimento deste projeto, por exemplo, Web Services e alguns dos seus detalhes, XML, DPWS e o framework .NET 3.5

Já no capítulo 3 abordaremos o DSB elaborado na dissertação de mestrado do Gustavo Medeiros de Araújo. Veremos sua estrutura principal, os principais meios de troca de mensagens entre dispositivos, e seu funcionamento em geral.

O capítulo 4 é dedicado à implementação do DSB feita com a tecnologia .NET, um detalhamento da estrutura, suas classes, as bibliotecas usadas, etc..

2. Revisão das Tecnologias

Este capítulo é dedicado a tratar sobre algumas das tecnologias que compõem projeto final, os assuntos são: SOA, Web Services, XML, SOAP, WSDL, DPWS e .NET Framework.

2.1. SOA

O SOA (*Service Oriented Architecture*) é uma arquitetura que permite que entidades de software utilizem serviços fornecidos por outras entidades. Com a constante automação de sistemas, é uma vertente que vem crescendo com muita força na internet, tanto academicamente como empresarialmente.

O SOA permite que sistemas se comuniquem por demanda ou periodicamente entre si, ajudando a evitar erros de chamada de procedimentos, auxiliando numa melhor distribuição de processamento entre provedores de serviço, evitando sobrecarga e podendo concatenar os procedimentos de um conjunto de provedores de serviço em um único provedor de serviço.

Um conceito muito importante que surge com o auxílio do SOA é a chamada "*cloud computing*", que significa "nuvem computacional", a qual utiliza em grande parte serviços hospedados como sendo seu alicerce central, podendo no caso oferecer além de serviços separados, conjuntos de serviços elaborados ao ponto de poder simular plataformas de execução de software ou mesmo simular servidores empresariais, podendo ter até local de armazenamento próprio na nuvem.

O *cloud computing* pode ser utilizado de três maneiras diferentes, que são:

Infrastructure-as-a-Service (IaaS): infra-estrutura como serviço que usa o exemplo utilizado anteriormente, no qual uma máquina virtual pode ser instanciada no provedor de serviços e o cliente, através de uma API disponibilizada pelo provedor de serviços, pode gerenciar sua máquina virtual.Podemos citar como uma infra-estrutura o serviço oferecido pela Amazon.com, por meio do qual é disponibilizado um servidor virtual.

Platform-as-a-Service (PaaS): plataforma como serviço, que disponibiliza aos desenvolvedores de software um local com ferramentas para desenvolver aplicações executando através de uma plataforma feita através de serviços. Nesta abordagem é comum utilizar *API´s* no dispositivo computacional que acessa a plataforma. Podemos citar como exemplo o *GoogleApps* (Google Apps, 2009). Porém também é preciso tomar cuidado com este tipo de desenvolvimento já que todo o código é feito dentro da plataforma, e é possível que o proprietário desta plataforma faça restrições quanto à retirada do código da plataforma.

Por último, porém não menos importante, temos o *Software-as-a-Service (SaaS)*: programa como serviço, que faz com que aplicações sejam disponibilizadas através da nuvem, bastando utilizar uma API para utilizar o programa, sendo que muitas vezes o próprio navegador é utilizado como uma API. Este é um modelo muito difundido atualmente. Como exemplo podemos citar aplicações Web como e-mails, sites de relacionamento, sites que disponibilizam músicas, entre muitos outros.

2.2.Web Services

Web Services é uma tecnologia que adota o paradigma SOA, utilizada para prover maior interoperabilidade e expansividade entre diferentes tipos de sistemas e plataformas. É possível utilizar *Web Services* de modo combinado, efetuando composições de serviços, conseguindo assim gerar informações com maior valor agregado.

Seu meio de transmissão de dados é via web através do protocolo SOAP, que permite a representação de dados utilizando XML, tanto para os parâmetros de entrada quanto para os de saída.

O objetivo principal de *Web Services* é a disponibilização de serviços via internet, de modo a utilizar o processamento de dados alheio para um desenvolvimento de

aplicações mais elaboradas e eficientes, sendo facilmente portados entre diferentes plataformas, mantendo um núcleo da aplicação sem necessidade de alterações.

O *Web Services* possuí meios para transmissão de dados pela rede, utilizando tanto parâmetros de entrada quanto retornos com os dados estruturados no padrão XML.

Muito úteis para o desenvolvimento de softwares que utilizam pouco recurso de hardware. Conseguem transportar grande parte do acesso aos dados ou processamento pesado para a máquina provedora de serviços, que poderia ser um servidor com muitos recursos de hardware provendo serviços através de uma rede, enquanto muitas estações de trabalho com grandes limitações de recursos de hardware utilizam um software cliente.

São identificados por um URI (*Universal Resource Identifier*), sendo que cada serviço disponível tem uma identificação distinta. Em cada serviço podem estar contidas diversas ações.

Cada ação ou método age de maneira independente e pode ou não ter um retorno para quem solicita a execução da ação. Se houver retorno por parte da ação requisitada, este pode ser associado aos mais diversos tipos de dados, pois pode se retornar tipos de objetos. Todos os retornos são codificados no formato XML e transferidos via protocolo SOAP.

2.2.1. XML

O XML (*Extensible Markup Language*) é uma forma de texto simples e muito flexível, derivada da SGML (ISO 8879). Originalmente desenvolvida para abordar o desafio da publicação eletrônica em larga escala, XML também está lidando com o importante papel na troca de dados variados na internet. (W3C)

É uma ferramenta para transportar informações independentemente de software ou hardware, pois o XML não passa de texto, ou seja, qualquer software que pode lidar com

texto consegue lidar com XML, aumentando drasticamente a quantidade de plataformas de software que podem suportá-lo.

É uma forma muito inteligente de agrupar dados, pois pode-se construir a estrutura de dados com atributos e outros dados encadeados, pode-se fazer estruturas do tipo árvore, usando tipos aninhados. A forma de organizar os dados é feita através de *tags*, representadas desta maneira:

Exemplo de XML:

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Exemplo 1

Desde 1998 o XML já é um padrão recomendado pela W3C (*World Wide Web* Consortium), que é um consórcio que desenvolve padrões para internet e que já criou mais de 110 padrões para a internet desde 1994, entre os quais estão padrões como: CSS, HTML, XHTML, etc.

2.2.2. SOAP

O SOAP é um protocolo feito para comunicação dentro de sistemas descentralizados e distribuídos. Trata-se de um protocolo baseado no XML que é dividido em três partes.

- O envelope que informa o framework usado para descrever a mensagem.
- Um conjunto de regras de codificação, que descrevem os tipos de dados, suas instâncias e em que aplicações são executadas.
- Uma convenção para representar a chamada de procedimentos remotos e suas respostas.

Os objetivos deste protocolo são: garantir uma maior expansividade e simplicidade. Através de uma generalização dos dados guardados dentro do SOAP é possível garantir que praticamente todos os tipos de dados serão suportados, desde que sejam informados do seu modelo de composição.

Segundo (Newcomer, 2002 , 3rd edition 2004) o SOAP define um padrão de mensagens para a troca de dados formatados em XML pela internet.

Para entender melhor o SOAP, segue um exemplo de chamada de um procedimento remoto:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
    <t:Transaction
      xmlns:t="some-URI"
      SOAP-ENV:mustUnderstand="1">
        5
    </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DEF</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo 2

No exemplo podemos notar a declaração de três tipos principais: o primeiro é o *envelope,* que é um campo obrigatório em uma mensagem SOAP e é quem engloba todos os dados da mensagem. O segundo é o *header,* que é um campo não obrigatório, e se aparecer têm que ser o primeiro elemento dentro do *envelope*. O terceiro elemento é

o *body,* um campo obrigatório que carrega as informações relacionadas à chamada de procedimentos remotos, atributos.

Segundo (Kühner, 2008) a mensagem inteira é contida em uma tag raíz chamada *Envelope*. Cada mensagem tem pelo menos uma tag chamada Header, que contém informações sobre endereços, id único da mensagem e o propósito da mensagem. Todas as informações sobre endereços contida dentro da tag Header é destinada a desacoplar o conteúdo do meio de transporte.

Uma curiosidade sobre esta tecnologia é que ela foi feita para apenas o envio de informações entre um suposto emissor e um receptor, porém não existe um modo de resposta dentro do SOAP. Por isso, foi convencionado que para receber a resposta seria necessário colocar um método no emissor no caso se o método retornar. O método se caracteriza por ter o mesmo nome do método chamado, porém com um "Response" concatenado no final, além de ser um método que não retorna nada.

Para fazer com que os elementos XML se comportem exatamente como o SOAP foi definido é necessário fazer a declaração dos *namespaces* de envelope e codificação.

Exemplos dos *namespaces* utilizados a seguir:

http://schemas.xmlsoap.org/soap/envelope/

http://schemas.xmlsoap.org/soap/encoding/

Exemplo 3

2.2.3. WSDL

A linguagem WSDL (*Web Services Description Language*) foi criada com o objetivo de se poder especificar os serviços definidos por um dispositivo computacional conectado a uma rede, através de um arquivo no formato XML. A WSDL de um serviço é apenas uma descrição, sem implementar qualquer dos procedimentos dos serviços, cujo papel é informar o desenvolvedor acerca dos serviços disponibilizados por um determinado Web Service.

Segundo (Newcomer, 2002 , 3rd edition 2004) o Web Services Description Language é um formato schema XML que define um espaço estendível para descrever as interfaces dos Web Services.

Exemplo de um documento WSDL 2.0:

Example 2-1. WSDL 2.0 Document for the GreatH Web Service (Initial Example)

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">

  <documentation>
    This document describes the GreatH Web service.  Additional
    application-level requirements for use of this service --
    beyond what WSDL 2.0 is able to describe -- are available
    at http://greath.example.com/2004/reservation-documentation.html
  </documentation>

  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://greath.example.com/2004/schemas/resSvc"
        xmlns="http://greath.example.com/2004/schemas/resSvc">

      <xs:element name="checkAvailability" type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element  name="checkInDate" type="xs:date"/>
          <xs:element  name="checkOutDate" type="xs:date"/>
          <xs:element  name="roomType" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
```

```
      <xs:element name="checkAvailabilityResponse" type="xs:double"/>

      <xs:element name="invalidDataError" type="xs:string"/>

    </xs:schema>
  </types>

  <interface  name = "reservationInterface" >

    <fault name = "invalidDataFault"
            element = "ghns:invalidDataError"/>

    <operation name="opCheckAvailability"
            pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/iri"
            wsdlx:safe = "true">
        <input messageLabel="In"
              element="ghns:checkAvailability" />
        <output messageLabel="Out"
              element="ghns:checkAvailabilityResponse" />
        <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
    </operation>

  </interface>

  <binding name="reservationSOAPBinding"
          interface="tns:reservationInterface"
          type="http://www.w3.org/ns/wsdl/soap"

wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

    <fault ref="tns:invalidDataFault"
      wsoap:code="soap:Sender"/>

    <operation ref="tns:opCheckAvailability"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

  </binding>

  <service name="reservationService"
       interface="tns:reservationInterface">

     <endpoint name="reservationEndpoint"
               binding="tns:reservationSOAPBinding"
               address ="http://greath.example.com/2004/reservation"/>

  </service>

</description>
```

Exemplo 4 (retirado do Web Site: *"http://www.w3.org/TR/2004/WD-wsdl20-primer-20041221/#basics-greath-scenario*")

Neste exemplo acima vemos que o primeiro elemento definido é uma tag XML (representada pelos sinais '<' e '>') informando que é um documento XML que utiliza a

codificação UTF8. Logo a seguir existe um elemento chamado "*description*" no qual todos os identificadores do WSDL: "*documentation*" que é uma documentação do Web Service, "*types*" são os tipos e onde são encontrados estes tipos, "interface" mostra as ações disponíveis do Web Service, "*binding*" faz a ligação entre a descrição e os serviços, "*service*" aponta onde o serviço está sendo executado.

2.3.DPWS

O DPWS (*Devices Profile for Web Services*) é uma especificação feita principalmente para dispositivos com limitações de recursos computacionais, a qual garante que sejam implementados um conjunto mínimo de funcionalidades.

Algumas das funcionalidades que se deve implementar são: permitir a troca de mensagens, poder descobrir novos serviços, descrever os serviços, e permitir troca de eventos. Pode-se dizer que um dos seus principais objetivos é poder fazer um ambiente de rede 'sem burocracias' para juntar-se à rede local. Funciona de forma parecida com a difundida tecnologia UPnP (*Universal Plug and Play*), fazendo com que os serviços disponibilizados via Web Services sejam rapidamente localizados e disponibilizados.

O DPWS foi desenvolvido em maio de 2004 pela empresa Microsoft em parceria com outras empresas e aprovado como um padrão OASIS em 30 de junho de 2009 na versão 1.1. Muitas empresas participaram do processo de padronização, entre elas estão: Microsoft, Canon, Schneider Electric, Odonata, IBM, CA, Novell, Software AG, Red Hat, Progress Software, WSO2, Fuji Xerox, Lexmark e Ricoh. (Padronização DPWS, 2008)

Segundo (Kühner, 2008) O Device Profile for Web Services descreve um padrão para procurar e se comunicar com dispositivos dentro de uma rede. A especificação define um sub conjunto da especificação Web Services, mas adiciona um mecanismo para que os clientes possam facilmente descobrir dispositivos.

DPWS funciona utilizando um padrão para representação de dispositivos e serviços, segundo o qual os chamados *Hosting Services* são os dispositivos propriamente ditos, enquanto os *Hosted services* são os serviços providos pelos dispositivos.

Para se construir uma aplicação no .NET Micro Framework 3.0 utilizando a especificação DPWS é necessário importar as bibliotecas:

- "MFWsStack.dll": Usada para representar um dispositivo, todos que usam a especificação têm que ter esta biblioteca referenciada.
- "MFDpwsDevice.dll": Usada para representar um provedor de serviços, necessária somente para os provedores de serviços.
- "MFDpwsClient.dll": Usada para representar um cliente de serviços; necessária somente para os clientes de serviços alheios.

É importante ressaltar que entre as bibliotecas "MFDpwsDevice" e "MFDpwsClient" citadas acima não existe uma inter-dependência; cada biblioteca pode ser referenciada individualmente sem problemas, porém não é garantido o funcionamento da aplicação sem a biblioteca "MFWsStack".

A descoberta de novos dispositivos é feita através da classe "Dpws.Client.DpwsClient", utilizando a biblioteca "MFDpwsClient". O método *Probe* é disparado em uma mensagem *multicast* e sua resposta vem com a descrição dos serviços oferecidos de forma *unicast*. Pode-se receber mais de uma resposta para uma única chamada do método *Probe*, sendo que as descrições recebidas nas respostas são armazenadas.

2.4. .NET

O .NET é uma plataforma para desenvolvimento de aplicações, construído pela empresa de software Microsoft, que apresenta suporte nativo para XML e Web Services. O .NET funciona como uma máquina virtual, onde todo código escrito sobre sua plataforma é convertido não diretamente em linguagem de máquina e sim num código intermediário.

De acordo com a máquina que vai reproduzir o código intermediário, a plataforma .NET se encarrega em converter esta linguagem intermediária em código de máquina, fazendo suas devidas modificações para melhor rendimento.

Uma das vantagens de se usar uma máquina virtual é a portabilidade do código, representada pela frase: "Write once, run anywhere". Essa frase foi o slogan de divulgação da plataforma do Java, feita pela empresa de software *Sun Microsystems*, que significa: "Escreva uma vez, execute em qualquer lugar".

É um conceito relativamente novo pois com a evolução constante dos hardwares a criação de uma plataforma que abstrai a máquina em que se está executando a aplicação, tornou o desenvolvimento menos dependente de arquiteturas específicas de hardware, e assim permitindo criar novas e mais complexas aplicações.

A plataforma .NET é composta principalmente de dois componentes a CLR (*Common Language Runtime*) e as bibliotecas de classes do .NET Framework, onde cada um desses componentes tem seu papel específico.

A CLR tem como papel o gerenciamento de memória das aplicações executadas sobre a plataforma, gerenciamento das aplicações, controle de execução das aplicações, acesso aos recursos de hardware, interpretação do código intermediário, e interoperabilidade entre diferentes tipos de hardware. Ou seja, o CLR tem o papel de máquina virtual na plataforma .NET, não diferenciando em que código o software foi escrito pois interpreta

código intermediário. Com isso, o programa pode ser escrito em Smalltalk.NET, C#.NET, VisualBasic.NET, F#.NET, entre outras linguagens suportadas pela plataforma.

Já a biblioteca de classes .NET funciona como uma caixa de ferramentas para o desenvolvedor, para não ser preciso “reinventar a roda”, por exemplo, estruturas básicas como tipos básicos de dados, tipos de interface gráfica, tipos de chamadas a procedimentos remotos, entre muitas outras estruturas estão disponíveis.

Existem diferentes distribuições do .NET *framework* também, como por exemplo o framework utilizado neste projeto que é chamado .NET *Micro framework*. É utilizado para componentes com pouco poder de processamento, memória, disco, recursos em geral. É justamente neste framework que o DPWS é utilizado, sendo parte das bibliotecas disponíveis, porém não se tratando de uma das bibliotecas nativas usadas por qualquer aplicação criada sobre ele.

3. Descrição do DSB

O Device Service Bus (DSB) é uma infra-estrutura de middleware que objetiva prover a integração de dispositivos heterogêneos em ambiente de computação ubíqua (Araújo, 2009).

O DSB é uma ferramenta que visa resolver limitações de comunicação entre tipos distintos de dispositivos computacionais.

O principal objetivo dessa infra-estrutura é criar um barramento de comunicação através do qual equipamentos com tecnologias de comunicação específicas possam cooperar uns com os outros, realizando serviços solicitados remotamente por seus pares (Araújo, 2009).

O DSB foi construído utilizando-se padrões internacionalmente conhecidos e divulgados, tais como XML, DPWS, Web Services, Framework J2SE, J2ME, entre muitos outros.

Como principal ferramenta ele utiliza a descoberta de novos membros na rede, tanto local quanto na própria internet, listando seus serviços e procedimentos oferecidos publicamente.

Através do DSB, cada membro novo na rede não precisa registrar-se manualmente em algum servidor para poder listar os serviços disponíveis na rede, basta simplesmente entrar na rede e estar utilizando o DSB para que uma lista de Serviços com suas diversas ações sejam disponibilizadas para que qualquer aplicação do seu dispositivo possa acessar.

O DSB foi desenvolvido para atuar em ambientes ubíquos, sendo que a qualquer momento os serviços disponibilizados por um dispositivo podem ficar inativos devido a desconexão do dispositivo.

Não há garantias de que os serviços disponibilizados estarão disponíveis na hora do uso ou sequer se retornarão uma chamada de método. Por isso, o DSB sempre mantém a lista de serviços disponíveis atualizada. Existe um ciclo de verificações constante, e sempre que um dispositivo que é responsável por um conjunto de serviços pára de responder, o DSB encarrega-se de remover o dispositivo e seus serviços da lista. Esse procedimento também ocorre com os novos dispositivos descobertos na rede, pois nestes ciclos, se algum dispositivo for reconhecido, ele será incorporado e seus devidos serviços serão adicionados à lista.

Cada dispositivo no DSB existe como um dispositivo virtual dentro do *middleware*, e a cada novo membro encontrado na rede, um novo dispositivo virtual é criado e adicionado na lista com uma identificação única. Quanto mais dispositivos na rede, maior a gama de serviços disponíveis, sendo que usuários com dispositivos com recursos computacionais limitados podem se aproveitar do DSB justamente para conseguir fazer tarefas que não seriam possíveis ou demorariam muito tempo em questão de segundos.

O DSB se comunica através do XML, e com o XML não há uma restrição de quantidade de dados ou tipos de dados que possam ser transferidos.

Um bom exemplo, ainda fictício, porém possível, de utilização do DSB seria um estudante chegando ao CTC, considerando os possíveis serviços a serem prestados pela rede da UFSC, que com seu celular acessa um programa que utiliza a lista de serviços do DSB para saber se vai chover nas próximas horas. Então ele verifica da posição onde ele está se existe algum provedor desta informação. Existindo o serviço o estudante faz a requisição e recebe em sua tela gratuitamente através da rede local a previsão do tempo. Seria possível também fazer milhares de tarefas mais complexas que demandem mais processamento.

A próxima figura mostra como um cliente encontra determinado serviço na rede, enquanto o *hosting service* utiliza o DSB:

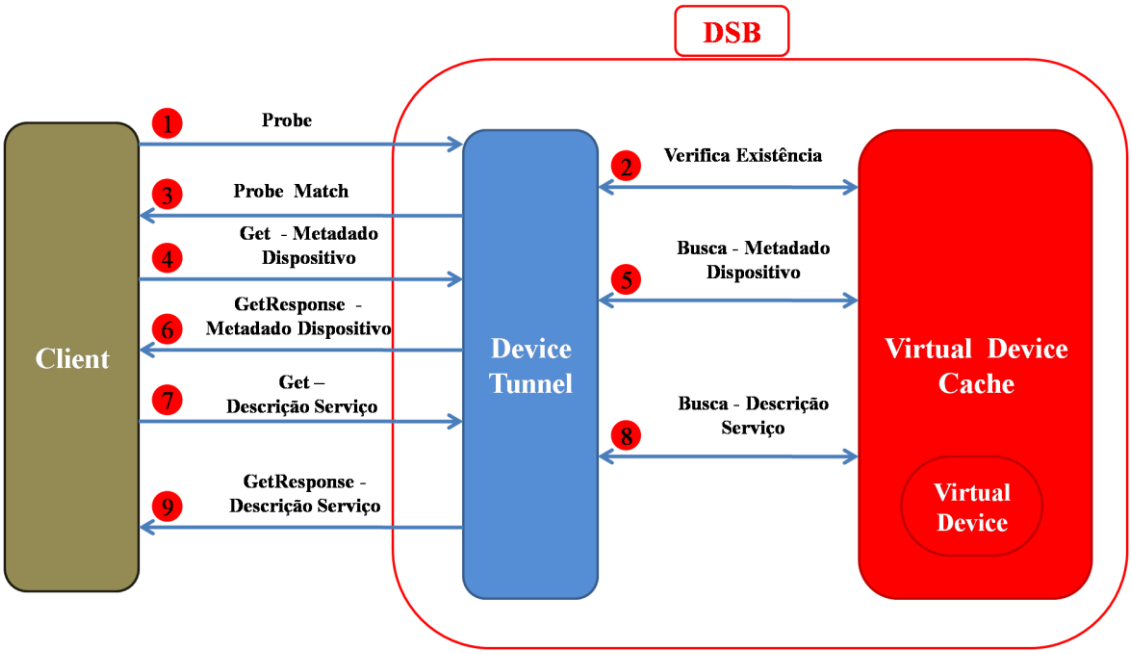

Figura 1 (Araújo, 2009)

1- *Probe*: Mensagem *multicast* enviada por um cliente em uma rede.

2- Verifica Existência: Um dispositivo DSB recebe esta mensagem de *Probe* e verifica se o serviço requerido pelo *Probe* existe nos serviços da lista de dispositivos.

3- *Probe Match*: Caso o DSB encontre o serviço em sua lista, é enviado ao emissor do *Probe* a mensagem de que o serviço procurado foi encontrado .

4- Get – Metadado Dispositivo: O cliente faz a requisição dos metadados do dispositivo que tem o serviço desejado.

5- Busca – Metadado Dispositivo: O *Device Tunnel* pesquisa no *Virtual Device Cache* que retorna os metadados requeridos.

6- *GetResponse* – Metadado Dispositivo: Os metadados requeridos são retornados ao cliente.

7- *Get* – Descrição Serviço: O Cliente faz a requisição do *WSDL* ao *DSB*.

8- Busca – Descrição Serviço: O *Device Tunnel* busca no *Virtual Devices*, que retorna o devido *WSDL* ao *Device Tunnel*.

9- *GetResponse* – Descrição Serviço: O Cliente recebe a descricao *WSDL* do *DSB* e agora consegue utilizar o servico requisitado.

3.1.Estrutura

A estrutura do DSB é composta de cinco tipos diferentes de componentes, são eles *Device Tunnel*, *Virtual Device Cache*, *Bridge*, *Converter* e *Tunnel Search Manager*.

O *Device Tunnel* funciona como um *hosting service* (provedor de serviços), o qual implementa a pilha de protocolos da especificação DPWS. Seu papel dentro do DSB é como se fosse um despachante, o qual encaminha as requisições aos devidos dispositivos e componentes.

O *Virtual Device Cache* funciona como uma lista de dispositivos provedores de serviços conectados à rede. Ele também é responsável por dispor interfaces DPWS para os dispositivos que não implementam DPWS. Cada dispositivo do *Virtual Device Cache* pode prover mais de um serviço distinto, e cada serviço pode conter uma lista de inúmeros procedimentos e eventos distintos. Para cada procedimento e evento é guardada a forma de invocação, o tipo requerido para cada um dos parâmetros de entrada e seu respectivo retorno, se existir.

O componente *Bridge* faz o papel de adicionar novos dispositivos na lista do *Virtual Device Cache*, através das mensagens definidas na especificação do DPWS (Especificação DPWS, 2006) *Hello* e *Bye,* que são enviadas via *multicast*. A *bridge* pode ser executada local ou remotamente. No caso da *bridge* remota, são usados dois componentes: a *BridgeStub* e a *BridgeSkeleton*.

A principal diferença entre estes dois componentes da *Bridge* Remota está exatamente no local onde são executados: a *BridgeStub* é executada localmente, enquanto a *BridgeSkeleton* é executada remotamente.

O *Converter* é um componente conversor das mensagens para determinado tipo de tecnologia: por exemplo, se dois ou mais dispositivos iguais se conectarem na rede eles utilizarão o mesmo *converter*. Já um dispositivo com arquitetura e troca de mensagens com padrões distintos utilizará um outro converter.

O DSB é baseado justamente na implementação de diversos tipos de *Converter*. Quanto maior o número de *Converters* criados, maior será a quantidade de dispositivos suportados, demandando mais desenvolvimento e assim expandindo o *middleware*.

O Converter tem como principais funções extrair os metadados dos dispositivos reais, fornecer a descrição dos seus serviços e encaminhar as chamadas de procedimentos para os dispositivos com os dados variando de acordo com suas diferentes formas de comunicação.

O componente *Tunnel Search Manager* realiza as buscas pelos dispositivos mantidos em *cache* pelo componente *Converter*. Primeiramente o *Tunnel Search Manager* faz a busca localmente procurando no *Virtual Device Cache* dentro do *middleware*. Caso não seja encontrado determinado serviço é feita uma busca remota enviando mensagens do tipo *multicast* na rede. Caso o serviço seja encontrado remotamente ele receberá via *unicast* os metadados e as descrições dos serviços providos por este determinado dispositivo. As mensagens então são encaminhadas para o *Converter* realizar seu papel de conversão e armazenamento dos metadados.

3.2.Funcionamento

O componente *Device Tunnel* é responsável por expor dispositivos e serviços encontrados na *Bridge* e *Converters*. O *Device Tunnel Manager* é responsável pela procura e pela obtenção de dados coletados através de mensagens *multicast* de busca.

A *Bridge* maneja os diferentes tipos de *Converters* e faz a conexão dos *Virtual Devices* com o núcleo da arquitetura. O *Converter* é responsável pela manipulação dos diferentes tipos de arquiteturas, tratando de questões específicas de cada tecnologia.

Justamente por tratar especificamente com a conversão dos metadados e fazer uma aproximação das trocas de mensagens do modelo original especificado por determinada tecnologia, que existe um *Converter* equivalente dentro do DSB. Sendo assim, mesmo que o provedor de serviços não possua o DSB executando em seu dispositivo, ele é capaz de interagir naturalmente com quem estiver executando o DSB.

4. DSB .NET

Desenvolvido sobre a plataforma .NET *Micro framework* 3.0, o DSB funciona como um conjunto de bibliotecas disponíveis para desenvolvedores, auxiliando uma troca de mensagens entre dispositivos com pouco poder de processamento de forma transparente, não se importando com detalhes de implementação dos diferentes tipos computacionais usados.

O DSB busca facilitar com o uso da especificação DPWS a descoberta de novos dispositivos e seus serviços públicos, porém o DSB faz com que aplicações que implementem o DPWS possam se comunicar com aplicações que não utilizem esta especificação, fazendo assim as necessárias conversões. De uma maneira mais direta, podemos dizer que o desenvolvedor pode utilizar as vantagens do DPWS sem a preocupação de adequar suas aplicações à tecnologia, bastando utilizar o DSB. O DSB, por sua vez, além de ter as vantagens da especificação DPWS, implementa outras vantagens como a comunicação com quaisquer outros dispositivos, sendo que a única barreira é o desenvolvimento de conversores para todos os distintos tipos de dispositivos.

Este trabalho é sobre a construção da parte interna do DSB, buscando implementar as vantagens da especificação DPWS, e possibilitando a troca de mensagens entre aplicações DPWS e não DPWS. Porém, não foram implementados os conversores, esta parte deve ser feita especificamente para cada dispositivo.

4.1.Estrutura

É estruturada em uma biblioteca, separada em cinco componentes principais: *Device Tunnel*, *Virtual Device Cache*, *Tunnel Search Manager*, *Converter* e *Bridge*.

Como o DSB também é dependente da tecnologia DPWS, muitas características se mantiveram, como por exemplo:

- Manter o dispositivo atual como um objeto estático, não sendo necessária a criação de instâncias desse objeto e assim mantendo-se como referência única dentro do DSB.

- Trabalhar com outros dispositivos DPWS de forma transparente, de modo que no outro dispositivo não seja necessária a utilização do DSB em suas bibliotecas.

- Manteve-se o esquema de biblioteca, no qual referenciando o DSB não é necessário referenciar o DPWS, pois ele é abstraído na implementação.

- Formas similares de entrar e sair de uma rede, através de mensagens *multicast Hello()* e *Bye()*.

O componente responsável pela ativação dos serviços é o *Device Tunnel.* Logo após a *Bridge* é adicionada como um serviço e ativada, e em seqüência são carregados dos diferentes tipos de *Converters* em uma lista.

Um exemplo da ativação de um *Device Tunnel* em uma rede seria o envio da mensagem *Hello() multicast*. A partir deste momento, para qualquer cliente que enviar a mensagem *Probe() multicast* para o mesmo grupo onde foi iniciado o serviço do *Device Tunnel*, ele responderá com uma mensagem *ProbeMatch() unicast*. O cliente, sabendo que existem provedores de serviço dentro da rede faz uma requisição *multicast,* efetuando uma busca por algum serviço pelo nome através do método *Resolve()*. Apenas os provedores de serviço que tiverem este determinado serviço disponível enviarão a mensagem *ResolveMatch()* ao cliente, restando apenas ao cliente pegar a descrição de todos serviços dos provedores escolhidos e fazer sua devida utilização. A dinâmica pode ser acompanhada neste fluxograma:

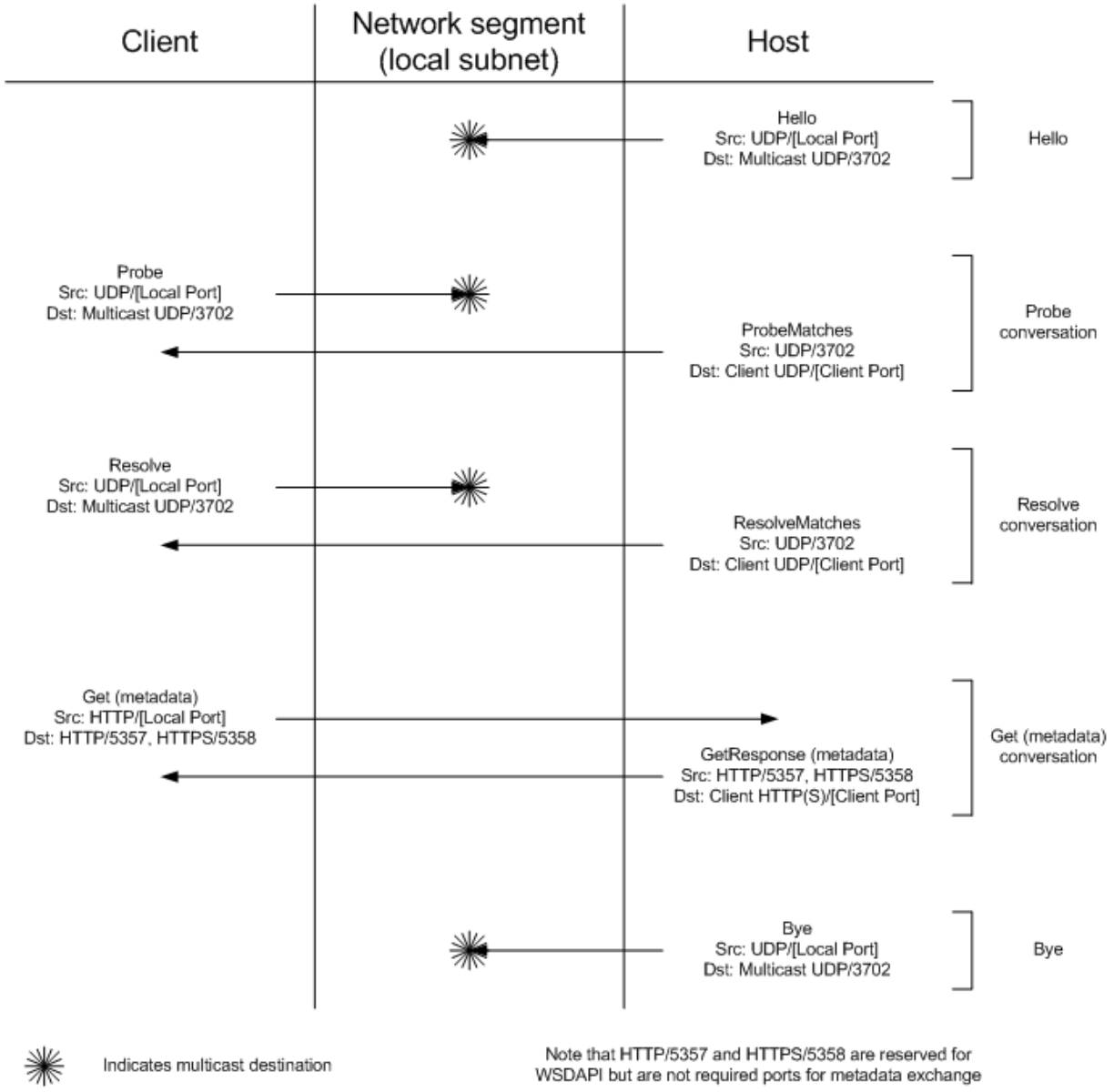

Figura 2 (Sobre Web Services on Devices, 2009)

Quando o DSB recebe uma mensagem *Resolve()* o componente responsável por buscar informações de disponibilidade de determinado serviço dentro do dispositivo atual é o Tunnel Search Manager, que fará uma busca somente local dentro da lista de componentes Virtual Devices, e se for encontrado uma ação dentro de algum serviço compatível, uma mensagem de ResolveMatch será enviada ao cliente. A busca é somente local pois a RemoteBridge não foi implementada.

4.2.Componentes

Esta sessão trata de explicar a função específica dos componentes, principais papéis desempenhados e também algumas relações entre componentes.

Device Tunnel: Na arquitetura desenvolvida este componente é o principal, responsável pela base da arquitetura, instanciando outros componentes e sendo a principal interface para quem faz buscas na rede. Ele será nosso *Hosting service*, o serviço principal.

Bridge: Atua como um gerente interno, adicionando e removendo dispositivos disponíveis dentro rede na lista de dispositivos *Virtual Device Cache*, também é responsável por unir grande parte dos componentes, sendo ele o componente que aguarda novas chamadas por parte dos *Converters* anunciando mensagens de entrada e saída de dispositivos na rede.

Tunnel Search Manager: Componente responsável por fazer buscas dentro da lista de dispositivos, caso não seja encontrado o serviço procurado ele trata de enviar uma mensagem de busca na rede.

Converter: O Converter funciona como um *plugin*, a cada novo converter criado é feita uma nova Thread para ficar na escuta de novos dispositivos. Transforma as informações recebidas para o modelo DPWS e faz o caminho oposto também.

Virtual Device: O *Virtual Device* é um componente que representa um dispositivo encontrado na rede, é um *Proxy* criado localmente para representar e poder invocar os eventos ou ações disponíveis. Uma lista desses dispositivos é armazenada e gerida pela *Bridge*.

4.3.Trabalho Realizado

O trabalho realizado aborda alguns pontos específicos do projeto em geral, não se limitando a apenas partes de implementação. Trata de explicar as idéias necessárias para o entendimento de como foi implementado.

O projeto em si foi feito todo dentro de uma solução, na forma de um projeto de biblioteca chamado DSB, que é justamente onde foi implementado o DSB.

O *framework* utilizado foi o *.NET Micro Framework 3.0* junto com as bibliotecas nativas do framework que suportam o DPWS: *MFWsStack*, *MFDpwsExtensions*, *MFDpwsDevice*, *MFDpwsClient*. Este conjunto de bibliotecas implementa as especificações DPWS.

O componente *Device Tunnel* implementou as funcionalidades principais de um provedor de serviços, chamando o método *start()* de um dispositivo, recebendo e repassando mensagens *multicast*.

O componente *Bridge* implementou uma classe dentro do *MFDpwsDevice* que representa um serviço disponível - um *HostedService* - fazendo papel de apenas um serviço para quem vê de fora, porém assumindo múltiplos papéis dentro do DSB.

O *Tunnel Search Manager* implementou parte do cliente do DPWS, realizando *Probe()* na rede em busca de serviços requeridos pelas aplicações usuárias do DSB, e procurando localmente se existe esse serviço.

O componente *Virtual Device* implementa a mesma classe da biblioteca *MFDpwsDevice* que a *Bridge*, porém aqui são armazenados os dispositivos encontrados na rede.

No caso do componente *Converter,* foram implementadas apenas as funcionalidades básicas para o correto funcionamento da tecnologia, pois este trata de questões específicas de cada dispositivo.

O funcionamento de um cliente quando um dispositivo com serviços é adicionado na rede é similar a um dispositivo que implementa apenas o DPWS e está disposto no exemplo:

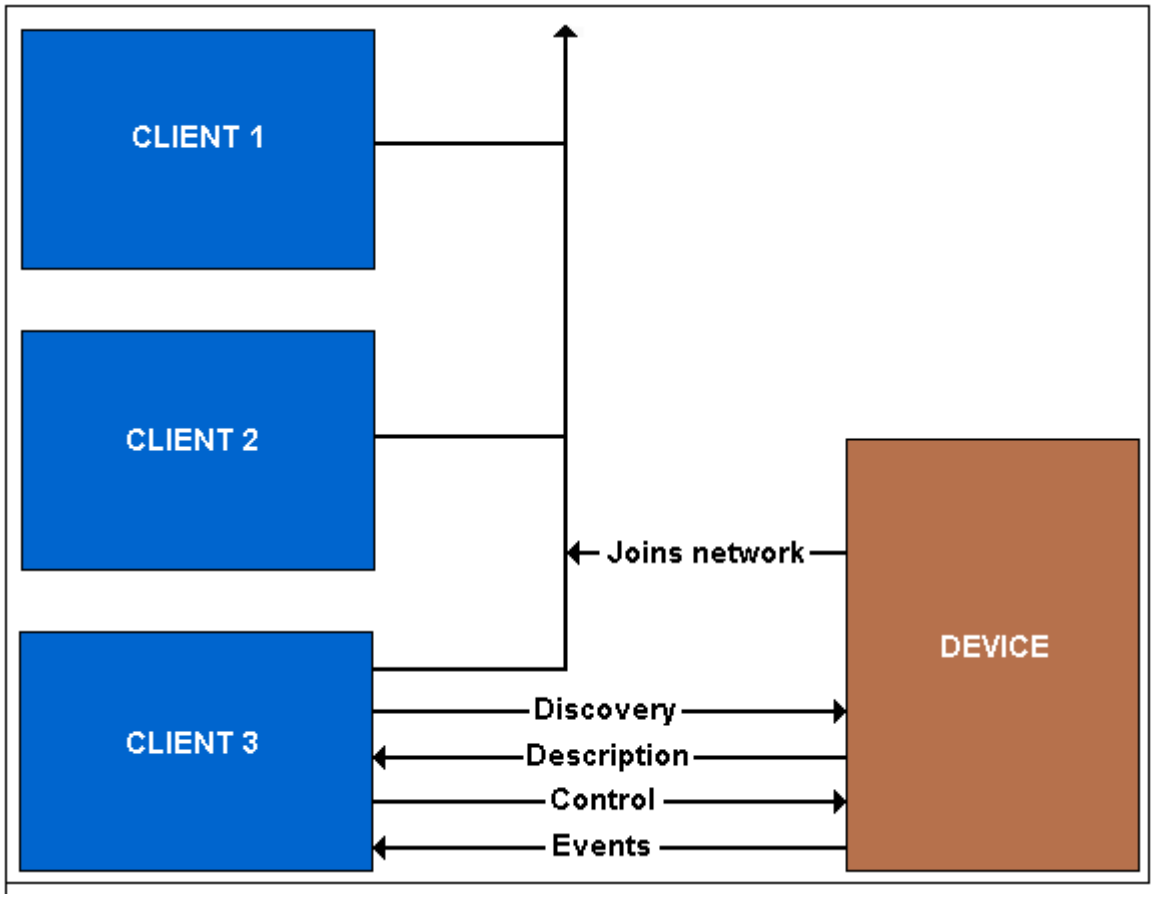

Figura 3 (Sobre Web Services on Devices, 2009)

Nesta imagem podemos acompanhar, quando um dispositivo provedor de serviços entra na rede, os clientes recebem uma mensagem do dispositivo. Os clientes enviam mensagens para descobrir as ações e eventos disponíveis no dispositivo e assim utilizam por demanda.

Como não foram implementados diversos tipos de conversores, não foi possível efetuar testes que pudessem comprovar uma utilização do *framework* com algum ganho de funcionalidades. Outro conceito muito importante porém deixado de lado neste trabalho foi o conceito de *bridge* remota. Com uma *bridge* remota seria possível suportar a seguinte situação: um dispositivo entra em uma determinada rede onde existem dispositivos que utilizam o DSB *bridge* remota, pega a lista de dispositivos do DSB *bridge* remota, podendo assim utilizar todos os serviços disponibilizados pelo outro dispositivo.

Neste caso a *bridge* remota seria outro componente. Faria o papel de repassar as informações dos dispositivos conectados ao mesmo. Assim sendo, quando um novo dispositivo entra na rede executando o DSB padrão, de forma automática ele receberia a lista de dispositivos da *bridge* remota e passaria a ser o detentor das informações de chamada das ações, e todos na rede poderiam utilizar o DSB para enxergar as ações repassadas pela *bridge* remota. Com isso, o novo dispositivo disponibilizaria serviços de outros dispositivos na rede.

A implementação ainda está sendo feita, por isso não foram disponibilizados gráficos de tempo de acesso a um determinado componente, nem foram feitos testes de compatibilidade na troca de mensagens com o DSB desenvolvido em Java. Porém espero que em breve já estejam disponibilizados estes resultados.

5. Conclusão

É um trabalho de grande complexidade, que teve de ser muito bem estudado, entender os conceitos aqui aplicados antes de qualquer implementação. Apesar de a biblioteca ficar com menos funcionalidades do que o esperado no início do trabalho, já foi um passo inicial para uma futura continuação deste projeto. Alguns dos objetivos puderam ser alcançados com sucesso como a migração de alguns dos componentes do núcleo do DSB.

Todo esforço despendido na criação deste projeto usando a plataforma .NET será de grande importância para um futuro próspero desta tecnologia, com muitos tipos de dispositivos suportados e maior número de plataformas disponíveis.

5.1. Trabalhos Futuros

A criação de novos tipos de componentes do tipo *Converter* dentro do .NET DSB será vital para a continuação deste projeto, já que não foi implementado por exemplo o *Converter* que trata das especificidades das tecnologias Bluetooth e RFID, que foram implementadas no DSB original.

Encontrar uma maneira de replicar os objetos estáticos dos provedores de serviços e transformá-los em objetos simples com a mesma funcionalidade.

Também será necessário criar as *BridgeSkeleton*, fazendo assim uma melhor aproximação do DSB original (Araújo, 2009), já que apenas a parte do núcleo do trabalho foi recriada.

6. Bibliografia

Araújo, G. M. (2009). *Dissertação de mestrado; Uma Infraestrutura para Integração entre Dispositivos Computacionais Heterogêneos Baseada na Especificação DPWS.* Florianópolis.

Artigo DPWS. (Fevereiro de 2006). Fonte: Specs Xml Soap.org: http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf

Definição de Cloud Computing. (Outubro de 2009). Fonte: SearchCloudComputing: http://searchcloudcomputing.techtarget.com/sDefinition/0,,sid201_gci1287881,00.html

Definição de WSDL. (Outubro de 2009). Fonte: World Wide Web Consortium: http://www.w3.org/TR/wsdl

Definição de XML. (Outubro de 2009). Fonte: W3Schools: http://www.w3schools.com/xml/xml_whatis.asp

Especificação DPWS. (Fevereiro de 2006). Fonte: Microsoft: http://specs.xmlsoap.org/ws/2006/02/devprof/

Google Apps. (10 de 2009). Acesso em 10 de 2009, disponível em Google Apps: http://www.google.com/apps/

Kühner, J. (2008). *Expert .NET Micro Framework.* United States of America: Apress.

Newcomer, E. (2002 , 3rd edition 2004). *Understanding Web Services.* Pearson Education.

Padronização DPWS. (Julho de 2008). Fonte: SODA.org: http://www.soda-itea.org/Events/1225727210.86.html

Sobre Web Services on Devices. (Outubro de 2009). Fonte: Microsoft: http://msdn.microsoft.com/en-us/library/aa385800%28VS.85%29.aspx

W3C. (s.d.). Fonte: W3C.org: http://www.w3.org/

Web Services. (Outubro de 2009). Fonte: World Wide Web Consortium: http://www.w3.org/2002/ws/

WSDL exemplo. (Outubro de 2009). Fonte: World Wide Web Consortium: http://www.w3.org/TR/2004/WD-wsdl20-primer-20041221/#basics-greath-scenario