Barramento de serviços para publicação de dispositivos na Web das Coisas

Carlos Henrique Fernandes Silva, Cássio V. S. Prazeres
Universidade Federal da Bahia – Departamento de Ciência da Computação
Av. Adhemar de Barros, Ondina, Salvador-BA, Brasil
{carloscte, prazeres}@dcc.ufba.br

RESUMO

Devido à evolução da Web, em paralelo ao surgimento de novas tecnologias, novas ideias e serviços têm surgido. Atualmente, as "coisas" fisicas também estão sendo incluídas na Web, possibilitando assim, o acesso aos dispositivos físicos a qualquer momento utilizando de protocolos e padrões já existentes. Essa possibilidade de interação com dispositivos físicos através da Web é denominada de Web das Coisas. Partindo desse princípio, é iminente o desenvolvimento de serviços que permitam controlar tais dispositivos, além de permitir e prover mecanismos para disponibilizar o acesso a eles via Web. Para tanto, devido à grande quantidade de coisas e dispositivos que podem ser acessados no nosso dia a dia, se faz necessária a existência de um ambiente de publicação e compartilhamento de serviços. Este trabalho tem por objetivo, através da implementação de um serviço e da utilização de um barramento de serviços, mostrar que é possível prover uma infraestrutura capaz de gerenciar a publicação de dispositivos físicos na Web das Coisas.

Palavras-Chave

Web das Coisas, Barramento, Publicação.

1. INTRODUÇÃO

A Web de Documentos, idealizada por Tim Berners-Lee et al. [3], passou por algumas evoluções desde sua implantação até a atualidade. Atualmente, a Web deixou de ser apenas páginas estáticas com documentos HTML como foi proposto inicialmente. Na Web atual, também chamada de Web 2.0 [1], pode-se desfrutar de aplicações mais interativas, com diversos serviços localizados em um só lugar (E-mails, Bate-papos, Bancos, etc. tudo dentro do próprio navegador) [2] e deixando de ser limitada à plataforma de computadores pessoais, podendo ser utilizada até mesmo por dispositivos móveis.

Com todo esse avanço da Web, novas ideias e funcionalidades também estão surgindo. Exemplos dessa evolução constante é a proposta da Web Semântica, conhecida como Web 3.0 e a Web dos Serviços e das Coisas (Web of Things - WoT).

Na Web Semântica, um dos objetivos é o de estender a Web de Documentos para a Web de Dados, fazendo com que os dados se relacionem entre si da mesma forma que os documentos atualmente já se relacionam. Para isso, utiliza-se de tecnologias como a Linked Data - Dados Interligados para fazer a ligação de recursos Web através de “links” semânticos, possibilitando aos usuários uma navegação de forma natural e intuitiva, seguindo esses “links”, por esses recursos, independentemente de interfaces de consulta específicas [3].

Das três evoluções da Web citadas anteriormente, a Web de Serviços é a que já está mais consolidada, dado que existem diversos Serviços Web disponibilizados na Internet. Os Serviços Web tem a funcionalidade de prover interoperabilidade entre aplicações na Web. Neste trabalho, utiliza-se Serviços Web para prover interoperabilidade entre dispositivos na Web das Coisas.

A Web das Coisas foi um paradigma de desenvolvimento de aplicações que surgiu inspirado na ideia da Internet das Coisas (Internet of Things - IoT) por volta de 1999 no MIT (Massachusetts Institute of Technology) [4]. Assim como a Web das Coisas, a Internet das Coisas consiste em ligar tudo à Internet, mas com algumas diferenças. Dentre essas diferenças, a principal está no modo de endereçar os dispositivos. A IoT consiste em fornecer um IP para cada dispositivo, efetuando assim o endereçamento na camada de rede. Já a WoT, o endereçamento dos dispositivos é feito na camada de aplicação, fornecendo uma URI para cada dispositivo [5]. O propósito da Web das Coisas consiste em ligar tudo (todas as coisas) à Web e formar uma rede em que cada objeto se comunique com outros objetos e gere informações para serem usadas de diversas formas, isso tudo utilizando dos protocolos já existentes na Web [4].

Com a utilização de URIs para endereçar cada um dos dispositivos físicos, a Web das Coisas tem utilizado bastante, junto ao protocolo HTTP, o estilo arquitetural REST (Representation State Transfer) [6]. O REST é aplicado em sistemas que são desenvolvidos seguindo uma arquitetura orientada a recursos, denominada ROA (Resource Oriented Architecture) [6]. Os sistemas seguindo esta arquitetura podem ser reutilizados e descritos sintaticamente. Esta descrição pode ser feita por um arquivo XML denominado WADL (Web Application Description Language), utilizado para descrever aplicações Web baseadas no protocolo HTTP [6]. Além disso, os princípios REST podem ser mapeados nos métodos do protocolo HTTP (GET, POST, UPDATE e DELETE). Dentre outras características, essas fizeram desse estilo arquitetural a melhor opção para construção de APIs Web para acesso a objetos do mundo real conectados a Web, e, por conta disso, utilizado para implementação deste trabalho [5].

A grande variedade de coisas, dispositivos do dia a dia, que podem ser acessadas utilizando a Web das Coisas, demanda por infraestruturas capazes de gerenciar a publicação desses dispositivos na Web.

Por tanto, este trabalho tem como objetivo a utilização de barramentos de serviços para gerenciar a publicação de dispositivos na Web das Coisas.

Para isso é necessário, modelar as funcionalidades dos dispositivos físicos como serviços na Web. Dessa maneira, é possível acessar tais funcionalidades a partir de URIs e utilizando os padrões da Web já existentes. É necessário também a

publicação desses dispositivos, modelados como serviços, em um barramento de serviços.

Partindo desses princípios, foi desenvolvido um protótipo com a finalidade de demonstrar a possível execução da proposta deste trabalho. Foi utilizado um barramento de serviços de código aberto, o MULE ESB (*Enterprise Service Bus*), para prover a publicação de aplicações. Como serviço, foi desenvolvida uma aplicação RESTful ("configurada" no padrão do barramento para a sua publicação no mesmo) que se comunica com dispositivos físicos através de um chip controlador Arduino (http://www.arduino.cc/).

Dessa forma, utilizando das técnicas citadas anteriormente, foi possível o desenvolvimento de uma aplicação capaz de gerenciar a publicação, através de um barramento de serviços (ESB).

2. INFRAESTRUTURA PROPOSTA

Nesta seção é explicitado o que foi feito para alcançar a resolução do problema. Tem como objetivo mostrar o fluxo, desde a aplicação utilizada para gerenciar os dispositivos físicos no chip do Arduino até a utilização da arquitetura REST para encapsular os dispositivos em Serviços Web.

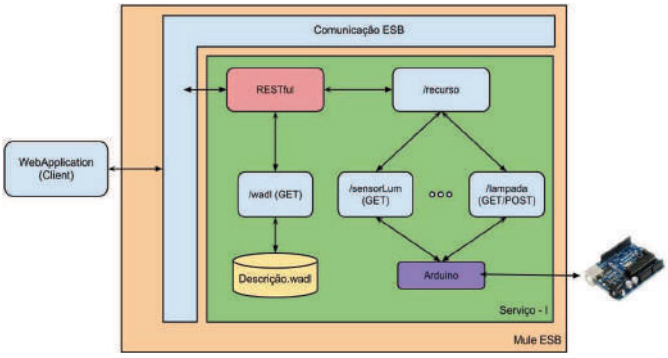

Figura 1. Modelo da infraestrutura desenvolvida

A Figura 1 representa o modelo referente à implementação deste trabalho. A partir de agora é feita uma análise detalhada deste modelo e dos componentes mais relevantes para o entendimento da parte prática que este trabalho se propõe.

2.1 Barramento de Serviço

Tudo se inicia com uma requisição de um usuário (cliente) ao barramento de serviço (ESB) que se encontra hospedado em um servidor na Web.

Este trabalho utiliza do barramento de serviços MuleESB. Neste barramento, a publicação e execução de aplicações é controlada por um arquivo XML de configuração.

Para a criação deste arquivo XML foi utilizado de uma ferramenta que auxilia, graficamente, todo o processo. O MuleStudio.

Dos diversos componentes oferecidos pelo barramento foi utilizado apenas um, o componente responsável por gerenciar a arquitetura REST. Esse componente utiliza de uma API já existente, o Jersey (http://jersey.java.net/). Dessa maneira, foi mais viável a utilização desse padrão de desenvolvimento para prover o acesso ao barramento e consequentemente à aplicação desenvolvida.

```
17  <flow name="projetoTccFlow1" doc:name="projetoTccFlow1">
18      <http:inbound-endpoint exchange-pattern="request-response"
19                             host="localhost" port="8081"
20                             path="servicoTcc"
21                             doc:name="HTTP"/>
22
23      <jersey:resources doc:name="REST">
24          <component class="org.classes.REST.Rest"/>
25          <component class="org.classes.REST.RecursosRecurso"/>
26          <component class="org.classes.REST.recursos.Lampada"/>
27          <component class="org.classes.REST.recursos.Sensor"/>
28      </jersey:resources>
29  </flow>
```

Figura 2. Trecho do arquivo XML de configuração MuleESB

Na Figura 2 está definido o único fluxo utilizado na implementação do trabalho. Nas linhas 18-22, explicitadas na figura, é definido o ponto de entrada (inbound-endpoint) do fluxo, que é basicamente a configuração do servidor e da localização de aplicação no próprio servidor. Já da linha 23 até a linha 28, está definido o componente primitivo do MuleESB responsável por toda arquitetura REST e indicação das classes da aplicação envolvidas nessa arquitetura.

Com essa configuração foi possível fazer com que, ao acessar o servidor do MuleESB, na porta especificada, passando as diretivas da aplicação, o barramento direcionasse o fluxo para a classe principal, e assim, desencadeasse o restante do processamento na própria aplicação. É importante deixar claro que o acesso ao servidor Mule é feito utilizando REST com os métodos do HTTP (GET, PUT, DELETE, UPDATE).

2.2 Arduino

Nesta seção é descrita a aplicação utilizada para programar o Arduino e fazer com que o mesmo se comunique com os dispositivos físicos a través de comandos "interpretados" pelo próprio Arduino.

É utilizado nesta aplicação o controle de 3 dispositivos físicos. São eles: 1 LED de cor verde, 1 LED de cor vermelha e 1 sensor de luminosidade. Os LEDs estão ligados à porta digital do Arduino e o sensor de luminosidade, como não é um dispositivo que fornece dados digitais, ligado à porta analógica.

A aplicação para programar o Arduino é escrita na linguagem de programação C e tem como objetivo ligar e desligar os LEDs, fornecer os status dos mesmos e capturar a luminosidade do ambiente.

Para conseguir obter esse controle de cada dispositivo físico de forma separada, foi criado um padrão de código a ser interpretado pelo Arduino a través da porta serial. O padrão consiste no seguinte: O primeiro caractere consiste no tipo de operação que o Arduino vai executar sobre o dispositivo. Caso seja "U" será uma operação de *update*, ou seja, será alterado o status do dispositivo (Só utilizado para os LEDs), caso seja "L", será uma operação de leitura, ou seja, será retornado para a aplicação REST o status do dispositivo (utilizado por qualquer um dos dispositivos citados). O segundo caractere é a porta física no Arduino em que se encontra ligado o dispositivo que se deseja alterar ou ler o status. E por fim, o terceiro caractere é utilizado apenas para as operações do tipo "U", pois será o valor que será enviado para o dispositivo físico em questão (como é utilizado apenas pela operação "U", consequentemente só é utilizado pelos LEDs).

Para exemplificar, suponha que queira acender o LED vermelho que se encontra conectado na porta 8 do Arduino. Para isso, será utilizada a seguinte combinação de caracteres: "U81". A operação selecionada é a *update*, o dispositivo é o que se encontra conectado na porta 8 do Arduino e o valor que irá passar para o

LED é o 1 (Ligar). Caso queira desligar o mesmo LED, a combinação de caracteres será a seguinte: U80. O "0" indica que o *update* será para desligar o LED. A Figura 3 mostra o trecho de código na linguagem C que trata essas *Strings*.

```
//Carrega no array a entrada de dados da porta serial
Serial.readBytes ( &entradaDados[0], 3 );

char operacao = entradaDados[0];
int recurso = entradaDados[1];
int novoValor = entradaDados[2];

switch(operacao){
  case 'L':

    if(recurso >= 8){// Porta digital
      Serial.write(digitalRead(recurso));
    }else{//Porta analógica
      Serial.write(analogRead(recurso));
    }

  break;

  case 'U':

    digitalWrite((int) recurso, novoValor);

  break;

}
```

Figura 3. Arduino utilizado para selecionar a operação.

2.3 Aplicação

2.3.1 Comunicação com Arduino

Esta primeira parte da aplicação contém as classes responsáveis pela comunicação com o Arduino e uma classe que tem como objetivo encapsular algumas constantes utilizadas na aplicação.

Foi criada uma classe chamada Arduino. Esta é utilizada como uma "caixa preta" para a comunicação com os dispositivos físicos. Ela é responsável desde a descoberta da porta serial, onde se encontra conectado o Arduino, até o envio e recebimento de comandos para o mesmo. Vale ressaltar que essa classe Arduino implementa a classe *SerialPortEventListener*, facilitando assim a comunicação com a porta serial. Foi implementada a função *serialEvent* da classe *SerialPortEventListener*. A implementação dessa função é extrema importância. É através dela que recebemos informações à partir do Arduino. A função *serialEvent* é chamada toda vez que houver alguma ocorrência de algum dado na porta serial onde houve a conexão, ou seja, para operações do tipo "L" (leitura) só foi possível através do uso desta função.

Essa função da Figura 4 é importante, porém, não é nesse código em que é feita a chamada para o envio nem o recebimento de dados. É utilizado apenas duas funções de forma explícita para a aplicação deste trabalho. A *getStatusDispositivo* e a *enviarComando*. Estão elas descritas na Figura 5. Essas são as duas únicas funções utilizadas de forma explícita, ou seja, chamadas diretamente das classes do grupo "REST".

A primeira função descrita na Figura 5 (*getStatusDispositivo*) consiste em enviar um comando de leitura (no padrão já citado na Seção 2.2) para o Arduino, aguardar uma resposta e retornar a resposta. Um detalhe importante dessa função é a utilização do método *this.pausaRapida*(). Esse método "pausa" a aplicação em um determinado tempo estabelecido nas contantes. Esse tempo é o tempo necessário para o Arduino processar o comando enviado, retornar a informação pela porta serial e haver a chamada implícita da função *serialEvent*, exibida na Figura 4. Quando há a chamada da função *serialEvent*, a mesma modifica o valor do atributo *respostaArduino*, tornando possível o retorno da resposta do Arduino à aplicação pela função *getStatusDispositivo*.

```
public synchronized void serialEvent (SerialPortEvent oEvent) {
    if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
        try {
            int myByte=input.read();
            int value = myByte & 0xff;//Transforma byte em inteiro

            this.respostaArduino = value;

        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

Figura 4. Implementação da função serialEvent

A segunda função descrita na Figura 5 (*enviarComando*) tem como propósito apenas enviar comandos para o Arduino. É utilizada de forma explícita apenas em operações de *update*, pois é enviado um comando sem a espera de uma resposta. E é utilizada de forma implícita na função *getStatusDispositivo*, pois essa envia um comando e aguarda uma resposta.

```
/**
 * Recupera o status de um determinado dispositivo
 * @param dispositivo - Porta do Arduino em que o dispositivo se
 *        encontra conectado
 * @return
 */
public String getStatusDispositivo(String dispositivo){
    this.enviarComando(Constantes.PREFIXO_LEITURA + dispositivo);
    this.pausaRapida();
    return String.valueOf(this.respostaArduino);
}
/**
 * Função utilizada para enviar comandos ao Arduino
 *
 * @param String
 */
public void enviarComando(String comando) {
    try {
        byte buff[] = new byte[221];
        buff = comando.getBytes();
        output.write(buff);
        output.flush();
    } catch (Exception e) {
        System.err.print(e.getMessage());
    }
}
```

Figura 5. Interface de comunicação com o Arduino.

Com a criação dessa classe, facilitada toda a parte de comunicação com os dispositivos físicos, pois quando há a necessidade de enviar ou ler dados dos dispositivos, basta instanciar um objeto desta classe e utilizar um dos métodos da Figura 5, a depender da necessidade.

2.3.2 Aplicação REST

Com toda a parte de comunicação com os dispositivos físicos implementada e encapsulada na classe Arduino, Agora será demonstrado como disponibilizar essa comunicação como um serviço REST. Nesta Seção é descrito como foi feito para gerar um serviço REST para disponibilizar esse acesso aos dispositivos físicos.

Essa aplicação foi desenvolvida com o intuito de ser uma API REST que fornece uma interface de comunicação com os dispositivos físicos. Levando em consideração um servidor local (*localhost*), serão mostrados alguns exemplos das funcionalidades providas por essa aplicação e, posteriormente, o quê e como foi feito para obter esses resultados.

Nesta seção, são mostrados e explicados alguns trechos de códigos da aplicação tidos como mais importantes e a arquitetura da aplicação para o alcance dos resultados. Como foi utilizado o Jersey para a implementação do REST, não existe uma classe inicial ou um ponto de partida da aplicação. Esse "ponto inicial" se destina à requisição feita ao barramento de serviço, e posteriormente de forma automatizada por esse, ao

encaminhamento do fluxo para a classe onde se localiza o recurso que deseja utilizar.

O recurso "*http://localhost:8081/servicoTcc/recurso/lampada/8*", por exemplo, é implementado conforme código apresentado na Figura 6. Nesse código, após a requisição feita ao barramento por esta URI, o próprio barramento processa a requisição, e o *Jersey* (embutido no barramento), já encaminha o fluxo direto para o método *getStatusRecursoLampada* mostrado na linha 36 Figura 6. Esse encaminhamento de forma automática é interpretado pelo Jersey necessitando apenas inserir no código o *@GET* (informando qual verbo do HTTP), *@Path* (informando o recebimento do parâmetro pela URI) e do *@Produces* (informando qual tipo de retorno será usado).

```
28    /**
29     * Utilizado para retornar o status da lâmpada (ligada ou desligada)
30     * @param idLampada
31     * @return
32     */
33    @GET
34    @Path("/{idLampada}")
35    @Produces("text/json")
36    public Recurso getStatusRecursoLampada(@PathParam("idLampada") String idLampada){
37
38        super.setNomeRecurso("Lâmpada");
39        Recurso lampada = super.getStatusRecurso(idLampada);
40
41        String status = lampada.getStatus();
42
43        if(status.equals(Constantes.OFF)){
44            lampada.setStatus(LAMP_DESLIGADA);
45        } else {
46            lampada.setStatus(LAMP_LIGADA);
47        }
48
49        return lampada;
50    }
```

Figura 6. Implementação do GET para recurso lâmpada

Dessa forma, fica mais claro o entendimento de como a requisição "*http://localhost:8081/servicoTcc/recurso/lampada/8*", após todo o processamento, executa o método *getStatusRecursoLampada* passando como parâmetro o ID da lâmpada desejada. O fluxo da requisição "*http://localhost:8081/servicoTcc/recurso/lampada/8*" foi direcionado para a execução do método *getStatusRecursoLampada* porque, primeiro que na requisição possui "/recurso/lampada", logo é direcionado para a classe Lampada e segundo porque é utilizado o método GET, passando o parâmetro 8 identificado na URI pelo "/8" no final.

Seguindo esses passos, foi criada toda a arquitetura REST do sistema possibilitando assim o encapsulamento da comunicação com o Arduino em serviços REST a serem disponibilizados no barramento de serviço.

3. CONSIDERAÇÕES FINAIS

No início deste Artigo foi proposto o desenvolvimento de uma infraestrutura que fosse capaz de gerenciar a publicação de dispositivos físicos na Web.

Partindo desta proposta, com a utilização de um barramento de serviço (*Enterprise Service Bus* - ESB), é possível prover um ambiente onde há a possibilidade de publicação de serviços na Web. Foi apresentado o modelo de arquitetura REST para o desenvolvimento dos serviços a serem publicados e a utilização de um chip controlador (Arduino) onde é possível o encapsulamento dos dispositivos físicos como Lâmpadas e Sensores em uma aplicação compilada e gravada no chip controlador.

Na implementação deste trabalho, foi possível a criação de uma infraestrutura para a publicação de serviços utilizando o MuleESB (barramento de serviços open-source), e o desenvolvimento de uma aplicação REST para o gerenciamento de dispositivos físicos. Para isso, foi necessário a criação de um arquivo XML de configuração utilizado pelo MuleESB para gerenciar a aplicação REST. Foi utilizado também uma API para Java (já incluída no próprio ESB e oferecido como um componente) para o desenvolvimento utilizando o padrão arquitetural REST. Para fazer a interligação entre a aplicação REST e o Arduino, foi criada uma classe onde foi encapsulada toda essa parte de comunicação, provendo assim uma interface para gerenciar essa troca de mensagens.

Está em fase de desenvolvimento uma maneira de auxiliar a descoberta automatizada desses serviços publicados no barramento. Para isso, foi utilizado o WADL para a descrição sintática da aplicação REST desenvolvida. Houve também a necessidade de modificar a estrutura padrão de diretórios da aplicação a ser publicada, para a inclusão do arquivo WADL, e a implementação de um recurso na própria aplicação REST para acessar e retornar o arquivo WADL. Dessa forma, foi possível disponibilizar a descrição sintática da aplicação para outros serviços ou clientes.

Assim, foi possível demonstrar a viabilidade da criação de infraestruturas capazes de gerenciar toda a parte de publicação, e futuramente de descoberta, de dispositivos físicos na Web das Coisas.

4. REFERÊNCIAS

[1] O'REILLY, T. What Is Web 2.0. Web Squared: Web 2.0 Five Years. v.1, n.1, p. , 2005.

[2] BRESSAN, R. T. Dilemas da rede: Web 2.0, conceitos, tecnologias e modificações. XXX Congresso Brasileiro de Ciências da Comunicação, Santos, n.1, p.1-13, 2007. Disponível em: <http://www.petfacom.ufjf.br/wordpress/arquivos/artigos/Artigo_2_Web_2.0.pdf> Acesso em: 12 mar. 2013.

[3] BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. Scientific American, Feature Article, v.1, n.1, p. , 2001.

[4] AYRES, M.; SALES, H. Internet das Coisas e Mobile Marketing: limites e possibilidades. Publicidade Digital: formatos e tendências da nova fronteira publicitária, Bahia, n.1, p. , 2010.

[5] DELICATO et al. Web das Coisas: Conectando Dispositivos Físicos ao Mundo Digital. Livro Texto de Minicursos do SBRC, Porto Alegre, v. , n.1, p.103-146, 2011.. In: Fabíola Gonçalves Pereira Greve; Ronaldo Alves Ferreira. (Org.). Disponível em: <http://www.nce.ufrj.br/labnet/pesquisa/cidadesinteligentes/minicurso-wot-final.pdf> Acesso em: 9 mar. 2013

[6] HADLEY, M. J. Web Application Description Language (WADL). Sun Microsystems Inc., , n.1, p.1 - 31, 2006. Disponível em: <http://wadl.java.net/wadl20061109.pdf> Acesso em: 11 mar. 2013