Web of Things: Automatic Publishing and Configuration of Devices

Nailton V. de Andrade Jr
Ciência da Computação
Universidade Federal da Bahia
Salvador, Bahia, Brasil
nailtonjr@dcc.ufba.br

Daniel Borges Bastos
Ciência da Computação
Universidade Federal da Bahia
Salvador, Bahia, Brasil
danielbb@dcc.ufba.br

Cássio V. S. Prazeres
Ciência da Computação
Universidade Federal da Bahia
Salvador, Bahia, Brasil
prazeres@dcc.ufba.br

ABSTRACT

The Web of Things proposes to make devices available by using Web standards and protocols. Several different devices, which can be connected to the Web of Things, demand efforts to implement specific services to deploy each of such devices. For that reason, this paper presents an approach to automatically publish and configure devices as Web of Things resources. Our approach presents models that implement device functionalities and uses such models to automatically generate Web Services for devices. This paper also presents: i) dynamic discovery and configuration of devices when connecting to a local network by using Zeroconf protocol; ii) automatic generation of applications for publishing devices on the Web of Things.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Services—*Web-based services*

Keywords

Web of Things; Configuration; Publishing; Discovery

1. INTRODUÇÃO

Com a evolução da computação embarcada é cada vez mais comum a presença de dispositivos físicos cotidianos com a capacidade de se conectar à Internet, entre eles eletrodomésticos, sensores e atuadores, automóveis e muitos outros. Segundo Guinard [11], tais dispositivos, chamados de "Coisas Inteligentes" (*Smart Things*), são dispositivos e objetos que apresentam capacidade de comunicação e processamento.

A Internet das Coisas (*Internet of Things* ou IoT) tem por objetivo conectar esses dispositivos inteligentes à Internet utilizando o protocolo IP (*Internet Protocol*) [5]. Entretanto, a heterogeneidade dos dispositivos, em relação ao formato de mensagens, por exemplo, torna não trivial a comunicação com e entre esses dispositivos. Dessa forma, a Web das Coisas (*Web of Things* ou WoT) visa estender a Web incorporando esses dispositivos à camada de aplicação utilizando o protocolo HTTP (*Hypertext Transfer Protocol*) da Internet.

A Web das Coisas será um ambiente onde todos os objetos físicos (eletrônicos ou não) do dia-a-dia, como edifícios, automóveis, mercadorias, matérias-primas, eletrodomésticos, sensores, dentre outros, se tornarão legíveis, identificáveis, endereçáveis e, ainda, controláveis utilizando serviços através da Web [11]. Isso permite a evolução de aplicações para a Web com uma vasta gama de novas oportunidades de negócio, tais como: suporte para a vida independente dos idosos [8], gestão eficiente de energia [1], gestão de ambientes inteligentes [19], gestão inteligente do tráfego [18], gestão de segurança pública e privada, gerenciamento eficiente de cadeias de suprimento e o monitoramento do meio ambiente [17], dentre outras.

Cicconi e Leibson [6] declararam que até o ano de 2015 cerca de 25 bilhões de dispositivos estejam conectados à Internet e que esse número deve crescer para 50 bilhões até o ano de 2020. Entretanto, nem todos os dispositivos são inteligentes, ou seja, existem dispositivos com recursos limitados, que não suportam o armazenamento e processamento de sistemas operacionais e aplicações inteligentes. Dessa forma, pensar na a Web das Coisas apenas para "coisas inteligentes" limitaria a sua abrangência e utilização.

A maioria dos dispositivos (mesmo os inteligentes) encontra-se atualmente desconectada da Web e mesmo da Internet, formando pequenas ilhas isoladas de softwares e interfaces proprietárias. Isso torna difícil a tarefa de integrar dispositivos a aplicações que poderiam ser reutilizadas para compor novos serviços [12]. Além disso, a grande diversidade de dispositivos, com diferentes formas de acesso e funcionalidades distintas, dificulta a sua disponibilização como recurso na Web.

Atualmente, para construir-se uma aplicação para a Web das Coisas, é necessário conhecimento específico de cada plataforma proprietária dos dispositivos envolvidos na aplicação. Segundo Guinard et al. [12], propostas de arquiteturas padronizadas, facilitam a integração dos dispositivos, entretanto, como não são totalmente compatíveis entre si, ainda necessitam da presença de especialistas. Dessa forma, essa diversidade e quantidade de coisas se conectando na Internet demanda soluções que possibilitem facilitar sua integração e, por conseguinte, disponibilização de suas funcionalidades como recurso na Web – tornando possível a Web das Coisas.

A realização da Internet das Coisas e, por consequência, da Web das Coisas está relacionada diretamente com so-

WebMedia'14, November 18–21, 2014, João Pessoa, Brazil.

http://dx.doi.org/10.1145/2664551.2664573.

luções para a implantação do protocolo IP em dispositivos embarcados. É importante que exista um mecanismo para encontrar e recuperar informações de dispositivos conectados à rede através de um endereço IP. O conjunto de especificações Zeroconf (*Zero Configuration Networking*) oferece métodos para descoberta de serviços em uma rede local e permite estabelecer uma rede sem a necessidade de configuração manual [3].

Este trabalho propõe uma solução baseada em modelos para configuração e publicação automáticas na Web das Coisas. A solução proposta utiliza o protocolo Zeroconf para a descoberta e configuração de dispositivos que se anunciam ao se conectarem em uma rede local. No momento em que o dispositivo se conecta na rede, o mesmo é identificado, e uma aplicação é gerada e publicada em um barramento de serviços, de forma dinâmica e automatizada. Essa aplicação permitirá o acesso às funcionalidades do dispositivo na Web via protocolo HTTP.

Nesse contexto, este trabalho propõe a utilização de dois tipos de modelos para a geração automática das aplicações: i) modelos que agrupam dispositivos por suas funcionalidades; ii) modelos de acesso para cada classe de dispositivo. Assim, quando um dispositivo se conectar a uma rede, a aplicação, que representa as suas funcionalidades, é gerada automaticamente a partir de seus modelos.

Baseado nesses modelos e no protocolo Zeroconf, este trabalho estende o barramento de serviços com uma nova funcionalidade: *Detecter*, que é responsável por buscar e resolver serviços registrados na rede por dispositivos físicos que se comunicam utilizando os mecanismos do Zeroconf. Este trabalho também apresenta o serviço *Generator*, externo ao barramento, responsável por gerar as aplicações, a partir dos modelos de funcionalidades e de acesso aos dispositivos.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta uma proposta de arquitetura para a Web das Coisas e principais conceitos relacionados. Em seguida, a Seção 3 descreve a proposta deste trabalho, utilizada para tratar o problema de configuração e publicação automáticas de dispositivos na Web das Coisas. A Seção 4 apresenta os experimentos realizados e a avaliação resultante desses experimentos. Na Seção 5 tem-se a discussão dos trabalhos relacionados e por fim as considerações finais e trabalhos futuros são apresentados na Seção 6.

2. WEB DAS COISAS

Atualmente, é cada vez mais comum a presença de dispositivos do dia-a-dia com a capacidade de se conectar à Internet. Segundo Duquennoy et al. [9], a conexão desses dispositivos à Internet não se refere a nenhuma estrutura de rede ou tecnologia específica, mas apenas à ideia de interconectar objetos, assim como é feito com computadores na Internet. Dado que os componentes de hardware e software dos dispositivos são muito heterogêneos, torna-se um grande desafio integrar esses dispositivos sem uma linguagem comum compreendida por todos eles [12].

Segundo Guinard [11], a Web é um exemplo de que é possível, utilizando um conjunto relativamente simples de padrões, tecnologias e protocolos abertos (HTTP, SOAP, HTML, XML, JSON, etc.), construir, no topo de uma infraestrutura de hardware e software heterogêneos, um sistema flexível e que preserva a eficiência e a escalabilidade. Assim, uma vez que muitos dispositivos estão sendo conectados à Internet, é natural usar os mesmos padrões e protocolos da Web como plataforma de integração. A utilização dos protocolos e padrões da Web para integrar dispositivos ou "coisas" do mundo real à Web foi denominada Web das Coisas.

A Web das Coisas permite que qualquer coisa seja utilizada como um recurso para a composição de aplicações Web projetadas para se relacionar com o mundo real. Com esse objetivo, padrões bem estabelecidos da Web podem ser reutilizados e adaptados. A arquitetura REST (*Representational State Transfer*) [10] preenche esses requisitos, uma vez que Serviços Web RESTful[1] utilizam os métodos do protocolo HTTP (GET, POST, PUT, DELETE) para aplicação dos princípios REST e URI's (*Uniform Resource Identifier*) para identificar e endereçar recursos unicamente.

Apesar de a Web prover protocolos e padrões que, por meio de serviços, permitem a troca de informações utilizando padrões como o XML (*extensible markup language*) e JSON (*Java Script Object Notation*), a grande variedade de coisas, dispositivos do dia-a-dia, que podem ser acessadas utilizando a Web das Coisas, possuem protocolos e formatos diferentes e muitas vezes proprietários [11]. Dessa forma, a Web das Coisas demanda uma infraestrutura capaz de gerenciar a configuração, publicação, descoberta, composição, utilização e compartilhamento desses dispositivos na Web.

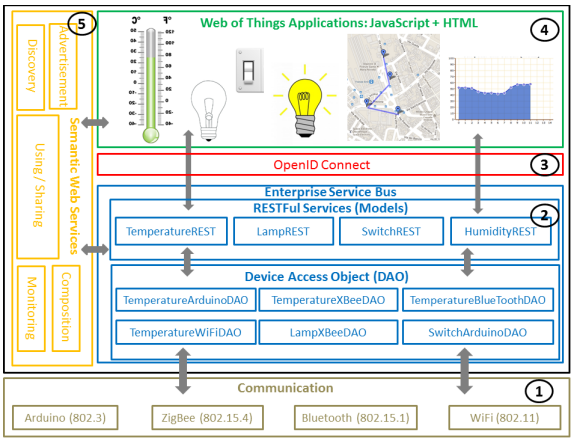

Figure 1: Arquitetura para a Web das Coisas.

A arquitetura apresentada na Figura 1 utiliza um barramento de serviços (*ESB - Enterprise Service Bus*) como infraestrutura base para configuração, publicação, descoberta, composição, monitoramento, utilização e compartilhamento para a Web das Coisas. Essa arquitetura pode ser dividida em 5 componentes maiores, que serão explicados a seguir: *Communication*; *Enterprise Service Bus*; *OpenID Connect*; *Web of Things Applications*; e *Semantic Web Services*.

Os resultados deste artigo, apresentados na Seção 3, estão diretamente relacionados aos componentes *Communication* e *Enterprise Service Bus*.

2.1 Communication

Conforme pode ser visto na Figura 1 (parte 1), "*Communication*" é o componente responsável por prover a comunicação com os dispositivos na rede. Nesse componente, deve ser possível descobrir, de forma automática, que um

[1]Serviços Web RESTful são serviços que utilizam padrões e protocolos da Web para implementar a arquitetura REST na Internet.

novo dispositivo se registrou na rede e descobrir, também automaticamente, qual o serviço provido pelo dispositivo.

Segundo Klauck and Kirsche [14], é importante que exista um mecanismo de descoberta, baseado no protocolo IP, e a habilidade de autoconfiguração para lidar com um grande número de coisas se conectando à Internet das Coisas. Neste trabalho, a solução utilizada (apresentada na Seção 3), para a descoberta automática e autoconfiguração, são os mecanismos do protocolo Zeroconf: *Multicast* DNS[2] (mDNS) e *DNS Service Discovery* (DNS-S).

Os dispositivos descobertos pelo componente "Communication" podem ser de dois tipos: burros e inteligentes. São considerados dispositivos burros, ou limitados, aqueles que não possuem capacidade de armazenamento e processamento: alguns sensores (por exemplo, temperatura e luminosidade) e atuadores (por exemplo, um semáforo de trânsito). Os dispositivos inteligentes, por sua vez, possuem poder de processamento e armazenamento, tal como tablets e smartphones.

Neste trabalho, estão sendo tratados apenas os dispositivos burros, que são descobertos, configurados e publicados no barramento – para posteriormente configurar suas permissões de compartilhamento (fora do escopo deste artigo).

Dispositivos inteligentes deverão decidir se querem participar do barramento. Por exemplo, um smartphone ou um tablet deve ter uma aplicação que: avisa ao seu proprietário que existe um barramento por perto; e pergunta se ele deseja disponibilizar acesso a algumas funcionalidades (por exemplo, GPS ou sensor de temperatura) do seu smartphone para outros usuários.

2.2 Enterprise Service Bus – ESB

Um barramento de serviços é middleware, com estilo arquitetural orientado a serviços, que visa fornecer recursos para facilitar a integração de aplicações por meio de serviços. Os principais componentes de um barramento de serviços são: ser um middleware orientado a mensagem, oferecer um container para serviços e fornecer uma camada de gerenciamento para configuração e monitoramento dos serviços e das mensagens.

Neste trabalho, uma vez que o componente "*Communication*" informou ao barramento de serviços (Figura 1, parte 2) que existe um novo dispositivo na rede e ainda qual é esse dispositivo, o barramento deve prover, automaticamente, uma forma desse dispositivo ser acessado na Web das Coisas – esse é o objetivo deste trabalho.

Para alcançar esse objetivo, são propostos neste artigo os modelos de acesso, baseados no protocolo de comunicação utilizado pelo dispositivo, para serem associados automaticamente aos dispositivos descobertos na rede. Esses modelos são chamados, conforme pode ser visto na Figura 1 (parte 2), de "Device Access Objects" (DAO), pois são componentes da arquitetura proposta responsáveis por se comunicar diretamente com os dispositivos físicos na rede.

Os DAOs são componentes altamente dependentes do dispositivo físico. Por exemplo, qualquer sensor de temperatura ZigBee[3] é acessado da mesma forma. Este trabalho utiliza um servidor de modelos que vai armazenar modelos de DAO para tipos de dispositivos conhecidos. Dessa forma, se o servidor de modelos já possui um DAO para um sensor de temperatura ZigBee, esse DAO vai servir para acesso a qualquer sensor de temperatura desse tipo que se registar na rede.

O mesmo servidor de modelos também vai prover os modelos de serviços RESTful (ver Figura 1, parte 2) que, em conjunto com os DAOs, vão possibilitar a disponibilização dos dispositivos na Web das Coisas. Note que os modelos de serviços RESTful devem ser independentes do dispositivo físico e dependentes das funcionalidades do dispositivo – um sensor de temperatura, independentemente de fabricante ou protocolo de comunicação tem a função de prover a temperatura de um ambiente. Em outras palavras, os modelos de serviços RESTful devem refletir as funcionalidades do dispositivo, enquanto que os DAOs refletem os protocolos de comunicação com cada dispositivo.

Essa abordagem, utilizando modelos DAO e modelos RESTful, possibilita a automatização da configuração e da publicação dos dispositivos no barramento (descrito na Seção 3), uma vez que não será necessário esforço de programação para tornar um dispositivo acessível na Web das Coisas.

2.3 OpenID Connect

O componente "*OpenID Connect*" deverá implementar, utilizando o protocolo *OpenID Connect*, dois modelos de autenticação e autorização para a Web das Coisas. O primeiro modelo foi desenvolvido em um trabalho [7] dos autores deste artigo e permite autenticar e autorizar proprietários dos dispositivos a terem acesso a seus dispositivos na Web das Coisas. O segundo modelo, que é um dos trabalhos em andamento relacionados a este artigo, permite que usuários tenham acesso a dispositivos de terceiros. Ou seja, permite que proprietários de dispositivos compartilhem o acesso aos seus dispositivos com outras pessoas.

2.4 Web of Things Applications

O componente "*Web of Things Applications*", apresentado na parte 4 da Figura 1, tem por objetivo propor modelos de visualização dos dispositivos e dos seus serviços oferecidos, na Web. Esses modelos podem ser reutilizados em diversas aplicações para a Web das coisas. O desenvolvedor de aplicações pode ser o próprio usuário final, desde que existam modelos para diversos tipos de dispositivos.

2.5 Semantic Web Services

Os Serviços Web Semânticos (*Semantic Web Services*) têm o objetivo de automatizar tarefas como descoberta, composição, execução e monitoramento de Serviços Web. Na arquitetura da Figura 1, o componente "*Semantic Web Services*" (parte 5) implementa algoritmos e métodos, baseados na descrição semântica dos serviços, de descoberta e composição automáticas para estender o barramento de serviços com semântica (fora do escopo deste artigo).

3. CONFIGURAÇÃO E PUBLICAÇÃO AUTOMÁTICAS DE DISPOSITIVOS

A variedade de dispositivos com a capacidade de se conectarem à Internet vem se expandido ao longo do tempo e as previsões em termos de números de dispositivos conectados são impressionantes para os próximos anos [6]. Ao passo que essa grande quantidade de dispositivos oferece as mais diversas oportunidades para os desenvolvedores e pesquisadores, configura também um desafio no que diz respeito à realização da Web das Coisas: integrar esses dispositivos

[2] *Domain Name System*
[3] http://www.zigbee.org/

heterogêneos na camada de aplicação. Dessa forma, esta seção apresenta a proposta deste artigo para configuração e publicação automáticas de dispositivos na Web das Coisas.

3.1 Modelos de Acesso e de Funcionalidades

Guinard [11] afirma que para desenvolver uma aplicação na Web das Coisas é preciso ter conhecimento específico sobre a plataforma, em geral proprietária, do dispositivo que se deseja integrar à Web. Em uma rede com um número grande de dispositivos, a configuração e publicação automáticas são questões essenciais que demandam soluções criativas e eficazes. A diversidade de formas de acesso às funcionalidades dificulta o processo de disponibilização automática desses dispositivos como recursos na Web das Coisas.

A proposta apresentada neste trabalho tem o objetivo de automatizar a configuração e a publicação de dispositivos como recursos na Web. Nessa direção, este artigo apresenta modelos, aqui chamados de modelos de acesso e modelos de funcionalidades, que na arquitetura apresentada na Figura 1, da Seção 2 deste artigo, são chamados de *DAO* e *REST*, respectivamente. Esses modelos, que, em conjunto, possibilitam a geração automática das aplicações para integrar esses dispositivos na Web das Coisas, são armazenados num servidor de modelos.

Os dispositivos físicos podem ser classificados de acordo com as funcionalidades providas. Dessa forma, o modelo de funcionalidades é um modelo de Serviço Web RESTful que depende apenas das funcionalidades que ele tem a oferecer – não depende da implementação do dispositivo. Por exemplo, o modelo *RESTModel* de semáforos de trânsito, apresentado na Figura 2, representa as funcionalidades de todos os dispositivos capazes de funcionar como um semáforo de trânsito. Qualquer coisa do mundo real pode ser classificada em um modelo predefinido para a Web das Coisas. Assim, para um dispositivo se conectar à infraestrutura aqui utilizada é preciso que seu modelo funcional esteja inserido no servidor de modelos.

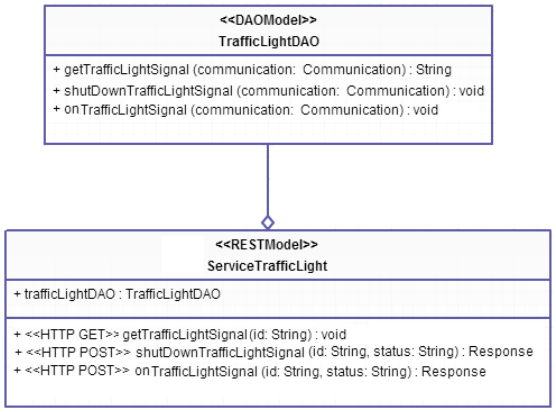

Figure 2: DAO e REST para semáforo de trânsito.

Além do modelo de funcionalidades, também é necessário que exista no servidor o modelo de acesso (DAO na na Figura 2), que tem por objetivo permitir a comunicação entre o dispositivo e o modelo de funcionalidades. É o modelo DAO que conhece os parâmetros de comunicação utilizados pelo dispositivo e sabe interpretar suas mensagens. Dessa forma, esse modelo é dependente da implementação do dispositivo e suas informações devem ser fornecidas por quem o planejou. A proposta é que o modelo DAO funcione como um "driver" para o dispositivo funcionar na Web das Coisas, da mesma forma que, por exemplo, uma impressora possui um driver para funcionar na rede ou em um sistema operacional específico.

3.2 Configuração e Publicação Automáticas

A Figura 3 apresenta uma visão geral da solução proposta neste trabalho. Foram desenvolvidos dois serviços principais com o objetivo da configuração e publicação automáticas dos dispositivos que se conectarem à rede. Esses serviços (ver Figura 3) são: o serviço *Detecter*, que está implantado no barramento de serviços; e o serviço *Generator* que está implantado em um servidor externo, responsável por gerar aplicações baseadas nos modelos do dispositivo.

Ainda é possível perceber na Figura 3 um servidor de modelos, cujo objetivo é armazenar os modelos de funcionalidades e os modelos de acesso que serão requisitados pelo serviço *Generator*, e um banco de dados, que armazena as informações colhidas do dispositivo e também as informações a respeito da aplicação gerada.

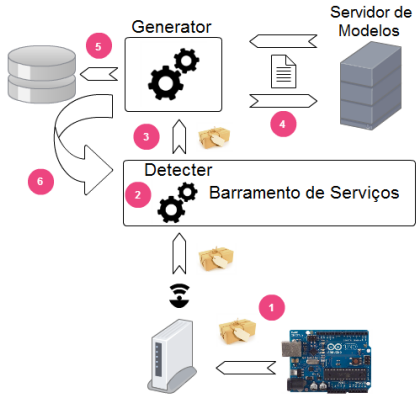

Figure 3: Visão geral da proposta.

A Figura 3 também fornece uma visão do fluxo seguido pelas aplicações no momento em que um novo dispositivo se conecta na rede. O Bonjour[4], utilizado neste trabalho, é uma implementação do protocolo Zeroconf e inclui, dentre outras coisas, a descoberta de serviços, o endereçamento e a resolução de nomes. Dessa forma, a utilização do Bonjour para descoberta de dispositivos evita a necessidade de configuração manual dos dispositivos, automatizando o processo de descoberta e resolução de serviços.

No passo 1 da Figura 3, o dispositivo se conecta na mesma rede em que o barramento de serviços está inserido. Nesse momento, o Bonjour implementado no dispositivo registra o serviço e se anuncia através de mensagens *multicast*. Uma vez tendo o dispositivo se anunciado, sua presença é percebida pelo serviço *Detecter* que também foi implementada utilizando API's baseadas no Bonjour. A função do serviço *Detecter* é perceber a entrada de novos dispositivos, ou seja, é responsável pela descoberta dos serviços como ilustrado no passo 2 da Figura 3.

Uma vez resolvido o dispositivo, o serviço *Detecter* colhe informações, como hostname, porta e informações extras

[4]http://www.apple.com/br/support/bonjour/

como o nome do modelo e a forma de comunicação do dispositivo. Com essas informações, o *Detecter* as envia para o serviço *Generator* (passo 3 da Figura 3), para que seja gerada uma nova aplicação. Tendo recebido o nome do modelo do dispositivo, é possível ao *Generator* requisitar ao servidor de modelos os arquivos necessários para construir a aplicação (passo 4 da Figura 3).

Por fim, o *Generator* salva as informações do dispositivo na base de dados (passo 5 da Figura 3), empacota e devolve a aplicação gerada (passo 6 da Figura 3) ao barramento de serviços, onde é implantada, tornando-a parte do barramento. Dessa forma, as funcionalidades do dispositivo estão publicadas na Web e podem ser acessadas via métodos HTTP contidos no modelo RESTful. A seguir, os serviços *Detecter* e *Generator* são descritos em detalhes.

3.2.1 Serviço *Detecter*

Para que ocorra a descoberta automática de dispositivos pela rede foi desenvolvido um serviço que: utiliza as funcionalidades do Zeroconf (via Bonjour); é executado no barramento de serviços; e monitora a rede e as alterações em sua topografia. O serviço que é responsável pela descoberta dinâmica e automática de dispositivos neste trabalho é chamado de *Detecter*, pois, suas responsabilidades são: detectar a entrada de novos dispositivos físicos na rede; colher suas informações; requisitar a geração de uma aplicação RESTful para este dispositivo; e, logo em seguida, implantar a aplicação gerada no barramento de serviços. A Figura 4 apresenta uma visão geral do serviço *Detecter*, que é dividido em três módulos principais: *Listener*, *Resolver* e *Manager*.

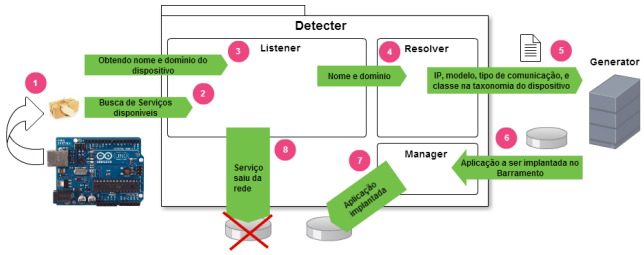

Figure 4: Visão geral do serviço *Detecter*.

O módulo *Listener* (ver Figura 4) é responsável pela busca de serviços disponíveis, assim, todo dispositivo físico que se conecte à rede será percebido nesse módulo do serviço (passos 1, 2 e 3 da Figura 4). Para possibilitar isso, o módulo *Listener* implementa os métodos do Bonjour: *serviceFound(...)* e *serviceLost(...)*, além do método *operationFailed(...)* da interface *BrowseListener*.

Ainda no serviço *Detecter*, o módulo *Resolver* (ver Figura 4) é responsável por resolver um serviço a partir do seu nome. Ou seja, uma vez que os serviços listados pelo *Listener* tenham sido determinados, seu nome e domínio devem ser passados para o *Resolver* (passo 4 da Figura 4), que implementa a interface *ResolveListener* em seus métodos *serviceResolved(...)* e *operationFailed(...)*. Para o DNS-SD, buscar serviços significa obter seus nomes e não endereços IP. Isso ocorre devido ao fato de que, em uma rede de conexão local ou DHCP, o IP de um dispositivo pode mudar com o tempo, de um dia para o outro, por exemplo. Dessa forma, as demais informações de um serviço, são obtidas, após resolvê-lo a partir de seu nome de instância, e repassadas para o *Generator* (passo 5 da Figura 4).

Ainda na Figura 4, é possível observar o módulo *Manager*, que tem por objetivo receber a aplicação gerada pelo *Generator* (passo 6 da Figura 4), através de uma chamada ao método POST do protocolo HTTP direcionado ao método *deployApp(...)* e implantá-la no barramento de serviços por meio do método *saveToFile(...)*.

Para desenvolver o serviço *Detecter*, que foi implantado no barramento de serviços, foi utilizado o framework *Jersey* o que simplifica o desenvolvimento de Serviços Web RESTful. O Jersey também gera automaticamente a descrição WADL (*Web Application Description Language*) do serviço desenvolvido. Neste trabalho, o documento WADL reflete o modelo REST do dispositivo (ver Seção 3.1), ou seja, descreve todas as funcionalidades do dispositivo. Essas funcionalidades são descritas no documento WADL por meio da descrição de todos os métodos HTTP possíveis para o dispositivo em questão.

3.2.2 Serviço *Generator*

O serviço *Generator*, ilustrado na Figura 5, é o responsável pela geração da aplicação que vai ser utilizada no barramento para publicar o dispositivo como recurso na Web das Coisas.

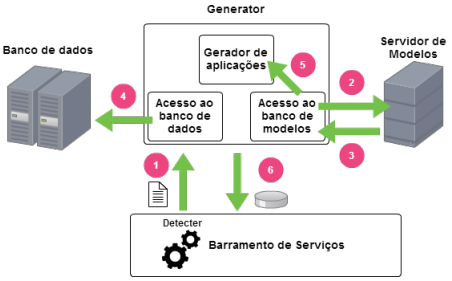

Figure 5: Visão geral do serviço *Generator*.

Para que o serviço *Generator* inicie a geração da aplicação, o serviço *Detecter* envia as informações através de uma chamada ao método POST do protocolo HTTP, cuja URI é mapeada no método *generateApp(...)* do *Generator*, tal como exibido no passo 1 da Figura 5.

Com essas informações, o serviço *Generator* requisita (passos 2 e 3 da Figura 5), por meio do método *requestModels(...)*, ao servidor de modelos, os arquivos dos modelos. Então, as informações do dispositivo são salvas na base de dados (passo 4 da Figura 5).

No passo 5 da Figura 5 ocorre de fato a geração da aplicação. Primeiro, a partir da escolha de uma porta randômica, o serviço *Generator* gera a URI que vai ser usada na aplicação e determina o nome da aplicação pelo padrão *<domínio>_<porta>_service_<NomeModelo>*. Alguns arquivos são comuns a todas as aplicações geradas, mas necessitam de certas modificações. Por exemplo, o documento WADL e o arquivo *application.properties*, que mantém certas definições que serão usadas pela aplicação gerada como, por exemplo, o tipo de comunicação. Dessa forma, o método *modifyWADL(...)* modifica o WADL, atribuindo, dentre outras coisas, a URI correta, e o método *savePropertiesFile(...)* modifica o arquivo *application.properties* para a aplicação em questão. Por fim, com todos os arquivos modificados, a

aplicação é empacotada e enviada de volta ao barramento (passo 6 da Figura 5).

O serviço *Generator* também foi desenvolvido com o framework *Jersey*, porém, diferente do serviço *Detecter*, foi implantado no servidor *Apache Tomcat*, externo ao barramento de serviços.

4. AVALIAÇÃO

Antes de apresentar os resultados da avaliação na Seção 4.2, a Seção 4.1 apresenta a configuração do experimento realizado e a descrição do protótipo de um semáforo de trânsito (modelos DAO e RESTful), que foi utilizado nos testes realizados neste trabalho.

4.1 Configuração do Experimento

Para o experimento foi utilizado um computador com 8GB de memória RAM e um processador Intel Core i5. O Barramento de Serviços (Mule ESB) e o Servidor Web (Apache TomCat), onde se encontram os serviços Detecter e Generator, respectivamente, foram instanciados no mesmo computador. Para configurar a rede local foi utilizado um roteador *D-LINK DIR 600*. O tempo de execução das funções mais importantes foi medido através da API JAMon[5], que é uma API Java de código livre e desenvolvida para monitorar o desempenho de aplicações Java.

Para avaliar os serviços *Detecter* e *Generator* foi implementado um protótipo de um semáforo de trânsito, utilizando-se de uma placa Arduino UNO composta pelo chip ATImega328 com 32 Kbytes de memória flash programável e 2 Kbytes de memória RAM e um placa Ethernet para permitir conexão via rede. A implementação do semáforo, com 3 *led's* coloridos (verde, amarelo e vermelho), pode ser visualizada através do esquema da Figura 6.

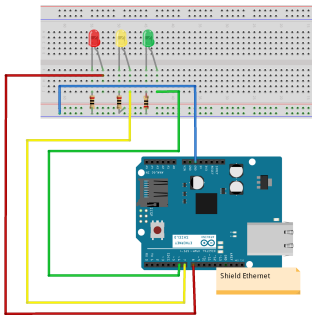

Figure 6: Protótipo Semáforo de Trânsito Arduino.

Conforme descrito na Seção 3.1 (Figura 2), o semáforo de trânsito é um atuator do tipo *TrafficLight* e seu modelo REST possui métodos GET e POST do protocolo HTTP. Esses métodos correspondem às funcionalidades: retornar o estado do semáforo, isto é, parado, siga ou atenção; ligar ou desligar o semáforo. As classes *TrafficLightDAO* e *ServiceTrafficLight* são, respectivamente, o modelo de acesso e o modelo REST de funcionalidades.

Para a aplicação ser gerada é necessário que os modelos do dispositivo estejam disponíveis no servidor de modelos. Qualquer modelo DAO é fornecido como um arquivo de biblioteca Java (.jar), sua classe deve ser nomeada no padrão <NomeModelo>DAO. O modelo TrafficLightDAO do semáforo Arduino fornece os métodos para obter o seu estado, ligá-lo e desligá-lo. Observa-se que toda função recebe um parâmetro do tipo *Communication*, a classe *Communication* é uma *Factory* que instancia a comunicação de acordo com o tipo salvo no arquivo de propriedades da aplicação gerada, que para o semáforo será Ethernet. A Figura 7 apresenta a interface Web de acesso à aplicação que foi gerada automaticamente para o protótipo do semáforo de trânsito.

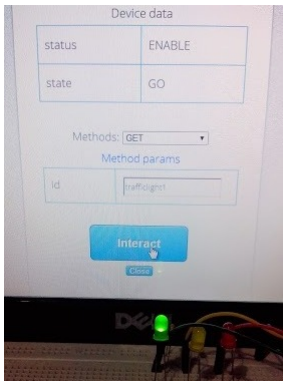

Figure 7: Aplicação gerada automaticamente.

Para que o protótipo do semáforo se registre e seja descoberto na rede, ele foi implementado utilizando o Boljour. Nesse protótipo, para implementar o Boljour no Arduino Ethernet foram utilizadas as bibliotecas *EthernetBonjour*, *EthernetDHCP* e *EthernetDNS* . Com essas bibliotecas, para registrar um serviço deve-se executar o método *addServiceRecord(...)* passando o nome do serviço, a porta do dispositivo em que o serviço está escutando na placa Arduino e o protocolo TCP/UDP de transporte. No caso do semáforo implementado, o nome do serviço registrado é *trafficLight_arduino._http*, comunicando-se através da porta 80 e usando o protocolo TCP.

4.2 Resultados da Avaliação

De acordo com a proposta deste trabalho, para que o dispositivo seja disponibilizado na Web das Coisas, duas etapas são necessárias: a configuração do serviço oferecido pelo dispositivo e a publicação da aplicação gerada.

A configuração é dividida em duas em duas etapas: a busca e a resolução. Neste trabalho, a busca é feita pela classe *Listener*, enquanto a resolução pela classe *Resolver*, ambas do serviço *Detecter*. No momento em que o dispositivo se conecta na rede, seu serviço é inscrito e uma mensagem mDNS é enviada, sendo percebida pelo *Listener* do *Detecter*. Uma vez que a busca depende da inscrição do dispositivo na rede e da mensagem mDSN enviada pelo próprio dispositivo, sendo o tempo de busca trivial, sua participação na performance total da solução é mínima, portanto não será avaliada. O mesmo pode ser dito em relação ao *Manager*, cuja funcionalidade é implantar a aplicação gerada no barramento de serviços.

Dessa forma, a performance da implementação é afetada principalmente pela resolução do serviço, realizada pela classe *Resolver* do serviço *Detecter*, instalado no barramento de serviços, e pela geração da aplicação no serviço *Generator*, hospedado no servidor *TomCat*. Essas duas etapas serão analisadas separadamente nas seções 4.2.1 e 4.2.2.

[5]http://jamonapi.sourceforge.net/

4.2.1 Resolução no *Detecter*

A classe *Resolver* (do serviço *Detecter*) é instanciada pela classe *Listener* assim que um novo serviço é percebido na rede. Nessa classe, uma chamada ao *DNSSD.resolve(...)* inicia a resolução do serviço, cujo nome e domínio foram recebidos. Dessa forma, foram medidos: o tempo de execução até que o método *serviceResolved(...)* seja disparado, denotando que o serviço foi resolvido com sucesso; e o tempo até que a mensagem tenha sido enviada pela chamada à função *callService(...)*. O tempo total da operação é a soma desses dois tempos.

Table 1: Tempo em ms de Execução Resolver.

Resolução	Preparação da Mensagem	Total
102,00	3,00	105,00
83,00	7,00	90,00
66,00	4,00	70,00
75,00	2,00	77,00
85,00	2,00	87,00
73,00	2,00	75,00
83,00	6,00	89,00
97,00	2,00	99,00
67,00	2,00	69,00
63,00	2,00	65,00
Média		
79,40	3,20	82,6

Foram realizadas 10 medições com o mesmo dispositivo semáforo Arduino, descrito anteriormente, para obter o tempo médio de execução, conforme Tabela 1. O tempo médio para a resolução do serviço é de 3,2 milissegundos, enquanto o tempo médio até o envio da mensagem com as informações é de 79,4 milissegundos. No total, a execução da classe *Resolver* leva em média 82.60 milissegundos. Esse tempo representa um tempo satisfatório para esse tipo de funcionalidade, uma vez que a descoberta dos dispositivos não deve impactar na infraestrutura da solução.

4.2.2 Geração de aplicação no *Generator*

O serviço *Generator* é responsável por gerar a aplicação que disponibiliza o dispositivo na Web das Coisas, a partir da requisição feita pelo serviço *Detecter*. Foram realizadas 10 medições para determinar o tempo médio de execução das funções do *Generator*. O tempo médio foi de 15.225,30 milissegundos, isto é, 15.2 segundos aproximadamente.

Table 2: Tempo em ms de Execução Generator.

Obter IP	Gerar Aplicação	Total
14.769,00	546,00	15.315,00
14.765,00	445,00	15.210,00
14.765,00	457,00	15.222,00
14.763,00	650,00	15.413,00
14.768,00	385,00	15.153,00
14.769,00	406,00	15.175,00
14.768,00	450,00	15.218,00
14.763,00	477,00	15.240,00
14.769,00	445,00	15.214,00
14.760,00	333,00	15.093,00
Média		
14.766,00	459,00	15.225,30

Conforme se obseva na Tabela 2, a maior parte desse tempo se deve a execução do código que retorna o endereço IP a partir do *hostname* do dispositivo. O tempo médio gasto para se obter o endereço IP é 14.766,00 milissegundos, ou seja, 14.8 segundos aproximadamente, enquanto que o tempo médio do resto das operações para gerar a aplicação leva apenas 459,00 milissegundos – total de 15,2 segundos.

Uma vez que o tempo para se gerar uma aplicação não depende das características do dispositivo, ou seja, do número de funcionalidades oferecidas, o tempo de 15,2 segundos representa uma vantagem quando comparado ao tempo que uma pessoa levaria para desenvolver uma aplicação manual para qualquer dispositivo.

5. TRABALHOS RELACIONADOS

Os requisitos e as dificuldades de projeto para utilização do protocolo Zeroconf em dispositivos inteligentes foram descritos por Jara et al. [13]. De acordo com esse trabalho, os três principais requisitos tecnológicos para a Web das Coisas são flexibilidade, escalabilidade e ubiquidade. Os autores propõem como solução para esses requisitos um conjunto de diretrizes chamado de lmDNS (*light-weight* mDNS), baseado em DNS-SD e mDSN, de forma otimizada e leve, que basicamente visa reduzir e comprimir certas informações utilizando algoritmos de compressão conhecidos.

Brama et al. [2] desenvolveram uma arquitetura de nós de sensores customizável e modular, de forma que as necessidades de comunicação entre os dispositivos sejam atendidas por uma abordagem aprimorada da Interface Periférica Serial (*Serial Peripheral Interface* - SPI) chamada flexSPI. As características do SPI relativas ao baixo consumo de energia e alta largura de banda foram mantidas, enquanto novos recursos foram habilitados tais como diagnóstico de conexão, descoberta de dispositivos e sincronização.

Mitsug et al. [15] apresentam uma proposta para facilitar o desenvolvimento de aplicações do tipo "*smart home*", compostas por dispositivos em uma rede de sensores. Os autores utilizam UPnP (*Universal Plug and Play*) e ONS (*Object Naming Service*) para automaticamente manter atualizada uma lista de dispositivos, e suas funcionalidades, que serão utilizados por desenvolvedores em aplicações.

Os três primeiros [13, 2, 15] trabalhos relacionados apresentam propostas de configuração automática de dispositivos.Nenhum deles integra a configuração automática a um barramento de serviços, de forma a estender as funcionalidades do barramento, que, neste trabalho, funciona como um gateway de objetos inteligentes para a Web das Coisas.

Choi et al. [4] apresentam uma proposta de configuração automática da interação entre dispositivos. Os autores utilizam de redes sociais para identificar relacionamentos entre dispositivos similares e prover interação entre eles. Para chegar ao ponto de identificar esses relacionamentos, os autores consideram que os dispositivos já estão publicados na Web das Coisas. Dessa forma, o trabalho de Choi et al. [4] pode ser complementar à proposta deste artigo, que visa automatizar a configuração e publicação dos dispositivos na Web das Coisas.

No que se refere ao gerenciamento de sensores e atuadores numa rede de sensores sem fio (WSN), Schor et al. [16] propuseram uma API, utilizando os princípios de arquitetura REST, para o monitoramento de edifícios inteligentes com o objetivo de reduzir o consumo de energia. Para tanto, os autores utilizaram a implementação do Zeroconf (também via Bonjour), que permite a descoberta dos serviços oferecidos pelos nós da rede. Os autores implementaram uma API

RESTful dentro dos dispositivos na rede, mesmo levando em consideração o uso de dispositivos com recursos limitados. Dessa forma, algumas técnicas com o objetivo de otimizar o uso dos dispositivos foram propostas. A solução final projetada consome 7.6 KBytes de RAM e ocupa 43 KBytes da ROM. Desse total, 0.7 Kbytes da RAM e 2Kbytes da ROM são usados pela API RESTful, o restante corresponde às implementações de protocolos de rede.

Quanto ao trabalho de Schor et al. [16], que mais se assemelha com o trabalho deste artigo, os autores implementam o serviço RESTful dentro do dispositivo, o que impede a utilização de dispositivos mais limitados. Por exemplo, o hardware utilizado no protótipo da Seção 4.1, possui 32 KB de memória FLASH (ROM) e 2 KB de RAM. A implementação, da parte que permite a comunicação Ethernet e o Bonjour no protótipo, ocupa cerca de 30 KB da ROM. Dessa forma, implementar o serviço RESTful dentro do dispositivo pode dificultar a utilização de dispositivos limitados. Além disso, a abordagem envolvendo modelos utilizada neste trabalho permite: que novos tipos de dispositivos sejam incluídos sem a necessidade de modificação na estrutura das aplicações; que os modelos sejam reutilizados em diversos dispositivos semelhantes.

6. CONSIDERAÇÕES FINAIS

A capacidade de configuração automática é importante em redes que possuem um grande número de sensores e atuadores conectados. Dadas as projeções sobre o crescimento no número de dispositivos se conectando à Internet nos próximos anos, fazem-se necessárias soluções que permitam a integração desses dispositivos na Web das Coisas.

Este trabalho propõe uma abordagem baseada no uso do protocolo Zeroconf e barramento de serviços para prover configuração e publicação automáticas de dispositivos como recursos na Web das Coisas.

Nessa direção, foram implementados dois serviços: i) o serviço Detecter que, através do Zeroconf, percebe a entrada de novos dispositivos na rede e coleta informações; ii) o serviço Generator, que gera a aplicação que disponibiliza as funcionalidades oferecidas pelo dispositivo em, implantando-a no barramento de serviços.

Para validar a proposta, um protótipo de um semáforo de trânsito foi desenvolvido, utilizando-se de led's que se alternam periodicamente, além de uso de bibliotecas específicas que permitem implementar as funcionalidades do Zeroconf, usando Ethernet, no dispositivo. O experimento realizado demonstrou o funcionamento da solução, tendo o semáforo sido disponibilizado para acesso via uma aplicação Web.

Um trabalho em andamento é a definição de uma metodologia para criação dos modelos, que permite a extensão dos modelos já existentes, podendo ser implementada como extensão em uma ferramenta de programação.

7. REFERENCES

[1] C. Beckel, W. Kleiminger, T. Staake, and S. Santini. Improving device-level electricity consumption breakdowns in private households using on/off events. *SIGBED*, 9(3):32–38, 2012.

[2] R. Brama, P. Tundo, A. Della Ducata, and A. Malvasi. An inter-device communication protocol for modular smart-objects. In *IEEE World Forum on Internet of Things*, pages 422–427, March 2014.

[3] S. Cheshire and M. Krochmal. Multicast DNS. Internet-draft, IETF Secretariat, Feb. 2011.

[4] J.-H. Choi, K. Kang, D. oh Kang, S. Yoo, and C. Bae. Towards zero-configuration in device collaboration using device sociality. In *IEEE World Forum on Internet of Things*, pages 417–421, March 2014.

[5] B. Christophe, M. Boussard, M. Lu, A. Pastor, and V. Toubiana. The web of things vision: Things as a service and interaction patterns. *Bell Labs Technical Journal*, 16(1):55–61, June 2011.

[6] J. Cocconi and S. Leibson. The internet of things. http://share.cisco.com/internet-of-things.html, January 2011.

[7] T. do Prado Filho, N. de Andrade Junior, F. de Farias, and C. Prazeres. Autenticacao e autorizacao para acesso a aplicacoes em um barramento de servicos para a web das coisas. In *III Workshop de Gestao de Identidades Digitais no SBSeg 2013*, pages 487–496, 2013.

[8] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier. The internet of things for ambient assisted living. In *Seventh International Conference on Information Technology: New Generations*, pages 804–809, April 2010.

[9] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle. The web of things: Interconnecting devices with high usability and performance. In *International Conference on Embedded Software and Systems*, pages 323–330, May 2009.

[10] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000. AAI9980887.

[11] D. Guinard. *A Web of Things Application Architecture – Integrating the Real-World into the Web*. Ph.d., ETH Zurich, 2011.

[12] D. Guinard, V. Trifa, and E. Wilde. A resource oriented architecture for the web of things. In *First International Conference on the Internet of Things*, pages 1–8, Nov 2010.

[13] A. Jara, P. Martinez-Julia, and A. Skarmeta. Light-weight multicast dns and dns-sd (lmdns-sd): Ipv6-based resource and service discovery for the web of things. In *6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 731–738, July 2012.

[14] R. Klauck and M. Kirsche. Bonjour contiki: A case study of a dns-based discovery service for the internet of things. In *11th International Conference on Ad-hoc, Mobile, and Wireless Networks*, pages 316–329, 2012.

[15] J. Mitsugi, Y. Sato, M. Ozawa, and S. Suzuki. An integrated device and service discovery with upnp and ons to facilitate the composition of smart home applications. In *IEEE World Forum on Internet of Things*, pages 400–404, March 2014.

[16] L. Schor, P. Sommer, and R. Wattenhofer. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 31–36, 2009.

[17] V. Tasic, T. Staake, T. Stiefmeier, V. Tiefenbeck, E. Fleisch, and G. Troster. Self-powered water meter for direct feedback. In *3rd International Conference on the Internet of Things*, pages 24–30, Oct 2012.

[18] C.-M. Vong, P.-K. Wong, Z.-Q. Ma, and K.-I. Wong. Application of rfid technology and the maximum spanning tree algorithm for solving vehicle emissions in cities on internet of things. In *IEEE World Forum on Internet of Things*, pages 347–352, March 2014.

[19] M. Weiss, A. Helfenstein, F. Mattern, and T. Staake. Leveraging smart meter data to recognize home appliances. In *IEEE International Conference on Pervasive Computing and Communications*, pages 190–197, March 2012.