UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Protótipo de uma central de controle para automação residencial compatível com o projeto IoTivity e controlada por aplicativo Android

Autor: Thiago Ghidoni Mantovan

Orientador: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2016

Thiago Ghidoni Mantovan

Protótipo de uma central de controle para automação residencial compatível com o projeto IoTivity e controlada por aplicativo Android

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica - Ênfase em Eletrônica

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos
2016

Mantovan, Thiago Ghidoni
M293p Protótipo de uma central de controle para automação residencial compatível com o projeto IoTivity e controlada por aplicativo Android / Thiago Ghidoni Mantovan; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2016.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2016.

1. Automação. 2. IoTivity. 3. Android. 4. Mobile. 5. Internet das Coisas. 6. Internet of things. I. Título.

FOLHA DE APROVAÇÃO

Nome: Thiago Ghidoni Mantovan

Título: "Protótipo de uma central de controle para automação residencial compatível com o projeto IoTivity e controlada por aplicativo Android"

Trabalho de Conclusão de Curso defendido e aprovado em 21/06/2016,

com NOTA 7,5 (SETE, CINCO), pela Comissão Julgadora:

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Dr. Dennis Brandão - (SEL/EESC/USP)

Prof. Dr. Maximiliam Luppe - (SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Dr. José Carlos de Melo Vieira Júnior

Dedicatória

Aos meus pais, Márcio e Selma, ao meu irmão, Matheus, ao meus avós, Marlene, Janitis e Waldemar e ao meu tio e tias, Dorival, Carla e Márcia, por me apoiarem e incentivarem, cada um a sua maneira.

Thiago Ghidoni Mantovan.

Agradecimentos

- À Deus, por tudo que já ocorreu em minha vida.
- Aos meus pais, Márcio e Selma, e ao meu irmão, Matheus, por todo o apoio em minhas decisões.
- Aos meu outros familiares, que colaboraram na minha formação tanto acadêmica quanto pessoal.
- Aos meus companheiros de classe, por compartilharem todas as alegrias e angústias.
- Ao Prof. Dr. Evandro Luis Linhari Rodrigues, pela orientação e conselhos dados a mim.
- À todos os professores que fizeram parte dos diferentes momentos de minha formação.
- Ao CNPq, por ter me proporcionado a experiência de realizar um intercâmbio.

Thiago Ghidoni Mantovan.

"Se algo é importante o suficiente, você deve tentar.
Mesmo que o provável resultado seja o fracaso."
Elon Musk

Resumo

Considerando o grande crescimento tanto das discussões quanto das aplicações envolvendo o tópico Internet das Coisas, esse projeto tem o objetivo de criar o protótipo de uma central de controle para um sistema de automação residencial controlado via comunicação sem fio por meio de um aplicativo Android. O diferencial desse projeto é a utilização do projeto IoTivity que é uma iniciativa criada pelo Open Internet Consortium e patrocinada pelas maiores empresas do ramo de tecnologia e que visa criar uma estrutura que possa proporcionar uma fácil integracão de diversas plataformas e sistemas operacionais com as diversas tecnologias de comunicação sem fio. O projeto apresenta um protótipo que oferece rápida resposta além de ser compatível com o projeto IoTivity e de fornecer a possibilidade do uso de outros dispositivos e sensores que ainda não seguem os padrões estabelecido pelo OIC. Esse protótipo é o primeiro passo para uma nova onda de dispositivos que seguirão os mesmos padrões e criarão uma situação na qual todos os dispositivos se comunicarão criando uma rede com controle descentralizado, aumentando a eficiência energética e promovendo um ambiente coeso que possa ser controlado por meio de um único aplicativo.

Palavras-Chave: Automação, IoTivity, Android, *Mobile*, Internet das Coisas

Abstract

Considering the huge growth in both discussions and applications involving the topic Internet of Things, this project aims to create a prototype of a home automation control center that is controlled wirelessly by an android app. The differential of this project is the inclusion of the IoTivity project which is an initiative created by the Open Internet Consortium, OIC, and sponsored by the major companies in the technology industry and aims to create a framework that provides an easy integration of different platforms and operational systems with the various wireless technologies. The project features a prototype that offers quick response in addition to being compatible with the IoTivity project and provide the possibility of using other devices and sensors that do not even follow the standards established by the OIC. This prototype is the first step to a new wave of devices that follow the same standards and create a situation in which all devices communicate creating a network with decentralized control, increasing energy efficiency and promoting a cohesive environment that can be controlled through a single application.

Keywords: Automation, IoTivity, Android, Mobile, Internet of Things.

Lista de Figuras

Lista de Tabelas

Siglas

IoT	*Internet of Things* - Internet das Coisas
OIC	Open Internet Consortium
OCF	Open Connectivity Foundation
npm	Node Package Manager
RFID	*Radio-Frequency Identification* - Identificação por Radio Frequência
IP	*Internet Protocol* - Protocolo Internet
HTTP	*HyperText Transfer Protocol* - Protocolo de Transferência de HiperTexto
HTML	*HyperText Markup Language* - Linguagem de Marcação de HiperTexto
CSS	*Cascade Style Sheets* - Folhas de estilo cascateadas
MQTT	*MQ Telemetry Transport* - Transporte de Telemetria MQ
SOC	*System on chip* - Sistema em chip
SOM	*System on module* - Sistema em módulo
SBC	*Single-board computer* - Computador em placa única
RISC	*Reduction Set Instruction Computer* - Computador com conjunto reduzido de instruções
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IDE	*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado
API	*Application Programming Interface* - Interface de Programcação de Aplicativos

Sumário

Capítulo 1

Introdução

Muito se tem discutido, recentemente, acerca da *Internet of Things* ou em português a Internet das Coisas no entanto, o conceito de dispositivos inteligentes conectados em uma rede já era aplicado muito antes do termo especifico ser cunhado. O primeiro registro de um dispositivo conectado é do começo da década de 1980 onde membros do departamento de Ciências da Computação da Universidade Carnegie-Mellon nos Estados Unidos conectaram uma máquina de refrigerantes ao computador do departamento e a partir dessa conexão tinham acesso a quantidade de garrafas de refrigerante contidas na máquina e se essas garrafas estavam geladas ou não [1].

Apesar da máquina de refrigerante ter sido o primeiro dispositivo conectado a uma rede, o aspecto internet ainda não estava presente uma vez que ainda não havia surgido a World Wide Web. Todavia, em 1990, John Romkey criou o primeiro dispositivo conectado à internet, uma torradeira que podia ser ligada e desligada através da internet [1].

Finalmente, em 1999 o termo foi cunhado por Kevin Ashton, diretor executivo do Aito-ID Center, ao realizar uma apresentação para a Procter e Gamble (P&G) na qual ele fazia uma ligação entre a ideia de chips RFID na produção da P&G e um dos assuntos que estava em alta no momento, a Internet. Em paralelo, Neil Gershenfeld do MIT Media Lab estava descrevendo algo semelhante em seu livro "When Things Start to Think"onde ele cita: 'Parece que o rápido crescimento da World Wide Web pode ter sido a faísca que está promovendo a verdadeira explosão, onde as coisas começam a usar a Internet' [1].

A discussão e o desenvolvimento nesse tópico foram muito grandes nos últimos 15 anos, ocorreram desde diversos relatórios, conferencias e publicações a diversas matérias em jornais e revistas de grande circulação. Entre 2008 e 2009, a Internet das Coisas nasceu, uma vez que existiam mais dispositivos do que pessoas conectadas à internet e em 2010 o número

de dispositivos já era de 12,5 bilhões enquanto a população mundial estava em 6,8 bilhões, ocorrendo pela primeira vez o fato de mais dispositivos do que pessoas estarem conectados à internet.

O surgimento de novas plataformas, novas normas, novos hardware e software aceleraram o crescimento da Internet das Coisas pois promoveram uma maior acessibilidade para pequenos desenvolvedores o que aumentou o interesse nesse tópico. Um exemplo dessas inovações que popularizaram o desenvolvimento foi o surgimento de placas de desenvolvimento de hardware sendo as mais famosas o Arduino e a Raspberry Pi.

Outro componente que não possuía tanta relevância a alguns anos atrás e que será um diferencial para manter o crescimento nesse tópico é o IPv6, pois esse novo protocolo disponibilizará uma gama de endereços quase infinita, para ser exato serão 2^{128} endereços, que será capaz de absorver todas as inovações que envolverão o acesso à internet, uma vez que o antigo protocolo, IPv4, atingiu o seu máximo número de endereços e seria um gargalo para o contínuo desenvolvimento de novas aplicações.

Juntamente com as inovações, diversas conferencias, relatórios e a entrada de grandes empresas no desenvolvimento e marketing desse tópico fizeram com que ele fosse adicionado em 2011 ao *Gartner Hype Cycle*, como poder ser observado na figura 1.1, na categoria *technology trigger*, que representa tecnologias em seus estágios iniciais com poucos produtos realmente prontos, com viabilidade comercial não comprovada porem com grande publicidade [2].

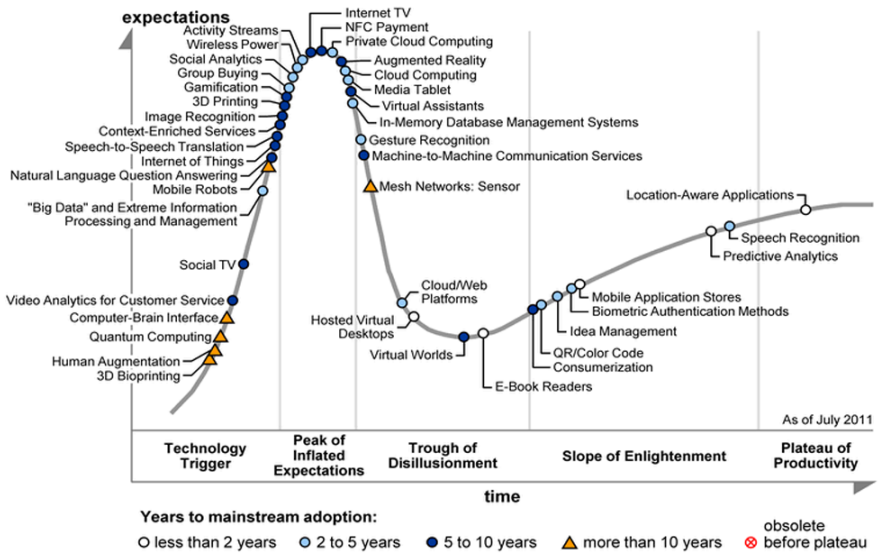

Figura 1.1: Gráfico Gartner Hype Cycle de 2011.

Em 2014, o tópico atingiu o ponto de *peak of inflated expectations*, como pode ser observado na figura 1.2, que representa o ponto de mais alta expectativa em relação ao assunto, o surgimento da divulgação de alguns primeiros sucessos e de fracassos também e da tomada de ações por parte de algumas empresas interessadas no assunto [3].

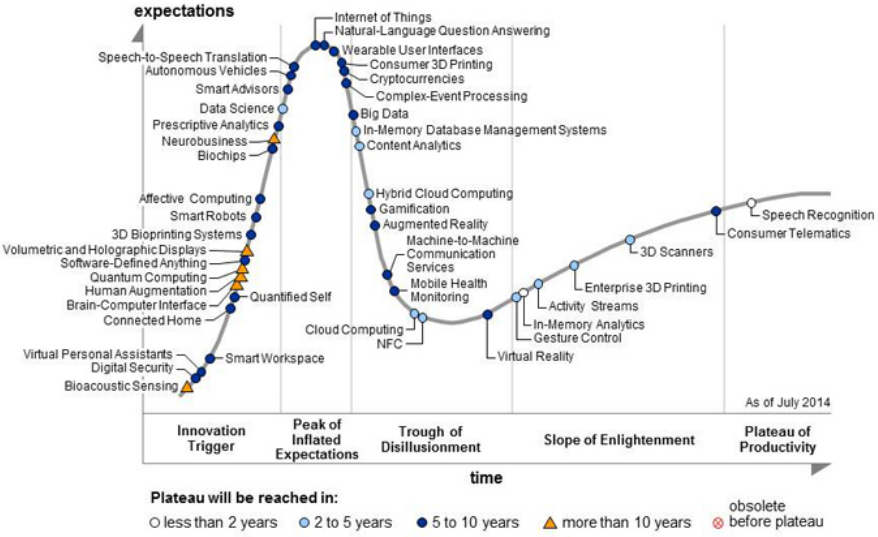

Figura 1.2: Gráfico Gartner Hype Cycle de 2014.

Com base nesse ciclo, a Internet das Coisas atingirá o ultimo nível, *plateau of productivity*, que representa o começo da adoção por grande parte da população, a viabilidade por parte do fornecedor fica mais clara e a grande aplicabilidade e importância da tecnologia oferecem um retorno mais concreto, entre 5 e 10 anos [3].

Mas qual será o próximo passo? Em um artigo publicado em 1999 na revista de negócios BusinessWeek que trazia ideias para o século XXI, uma delas era de que o planeta seria envolvido por um tipo de pele eletrônica. Sendo que essa pele seria constituída por milhões de dispositivos de medição: termostatos, detectores de poluição, câmeras, microfones e diversos outros sensores. E esses dispositivos iriam testar e monitorar as cidades, os animais, a atmosfera, as rodovias, a nossa comunicação e até nossos corpos [4].

Analisando o artigo e observando os avanços obtidos nessa área nos últimos anos podemos ver que estamos seguindo o caminho citado e que a evolução disso é o fim da necessidade de uma central que controle os dispositivos pois eles serão capazes de trocar informações entre eles e realizar ações autônomas com base nas preferencias já observadas dos usuários.

E é baseado nessa visão e no desenvolvimento que está ocorrendo no momento, que esse projeto tem o intuito de ser um passo inicial para esse novo cenário que está sendo imaginado

e construído.

1.1 Motivação

A motivação deste trabalho se deu após a realização da disciplina SEL0373 - Projeto de Sistemas Digitais na qual foi desenvolvido um sistema inicial de automação residencial utilizando a placa Intel Galileo e alguns sensores de modo a simular o funcionamento de diversos dispositivos em uma residência.

Além da disciplina cursada, outro fator que influenciou nessa decisão é a magnitude que o assunto internet das coisas tomou nos últimos anos, a diversa gama de aplicações nas quais ele pode ser usado e uma excelente perspectiva de crescimento desse setor nos próximos anos.

1.2 Objetivo

O objetivo desse trabalho é criar um protótipo de uma central de controle para automação residencial que esteja em conformidade com os padrões definidos para a Internet das Coisas pelo Open Internet Consortium, que esteja preparado para integrar dispositivos que utilizem o projeto IoTivity, além de oferecer a possibilidade do uso de dispositivos independentes produzidos especialmente para esse sistema.

Para mensurar se o objetivo foi alcançado, será realizado um teste de latência do sistema e um teste com exemplos fornecidos pelo projeto IoTivity para confirmar a integração das funcionalidades oferecidas por essa iniciativa.

1.3 Justificativa

A justificativa deste trabalho é criar uma aplicação utilizando o que há de mais avançado no tópico Internet of Things, que esteja dentro das especificações que estão sendo criadas pelo OIC e que gere conhecimento para trabalho futuros próprios ou de terceiros.

1.4 Organização do Trabalho

Este trabalho está distribuído em sete capítulos, incluindo esta introdução, dispostos conforme a descrição que segue:

Capítulo 2: Descreve os conceitos e ferramentas utilizados para a produção deste trabalho

Capítulo 3: Discorre sobre os materiais e métodos utilizados durante o desenvolvimento deste projeto

Capítulo 4: Apresenta os resultados obtidos e discute o grau de satisfação atingido além da relevância do projeto

Capítulo 5: Conclui sobre o trabalho ponderando os avanços alcançados e os obstáculos encontrados além de apresentar possíveis modificações ou evoluções para trabalhos futuros

Capítulo 2

Embasamento Teórico

O projeto se baseia em três pilares: sistemas embarcados, protocolos de comunicação e o projeto Iotivity. Uma visão geral sobre esses tópicos é providenciada a seguir.

2.1 Sistemas Embarcados

Um sistema embarcado é um sistema baseado em um microprocessador que é construído para realizar uma ou diversas tarefas e que não é projetado para que o usuário final possa programá-lo. Geralmente, o uso de sistemas embarcados tem como objetivo tanto de fornecer dados quanto de realizar tarefas previamente programadas instantaneamente visto que foram projetados para realizar tais tarefas mais eficientemente que outros sistemas de uso mais generalizado [5].

Uma evolução dos sistemas embarcados mais convencionais são os *single-board computers*, ou SBCs, que significam computadores de placa única, que possuem além do microprocessador, a memória, entradas e saídas e outras características requeridas para um computador funcional, tudo em um única placa. Alguns exemplos de SBC são a Raspberry Pi [6], a BeagleBone Black [7], os Arduinos [8], a Intel Galileo [9].

Com a melhora na processo de fabricação de chips integrados que proporcionou aumentar muita a densidade de componentes presentes em um chip, surgiram os *systems on a module*, SOMs e os *systems on chip*, SOCs, que significam respectivamente sistemas em um módulo e sistema em um chip. Esses sistemas oferecem desde conexões Wi-Fi e Bluetooth a até entrada e saídas digitais e analógicas. Exemplos desses sistemas são o Intel Edison [10], ESP8266 [11], as placas Colibri da Toradex [12].

Os sistema embarcados variam desde dispositivos portáteis como relógios digitais e toca-

dores de MP3, a até grandes instalações como semáforos, controladores de fábricas e outros muito complexos como veículos, aparelho de ressonância magnética e outros. A complexidade pode variar desde um único microcontrolador a até unidades que necessitem de de vários microcontroladores.

A comunicação com o mundo físico é feita por meio de periféricos como interfaces de comunicação serial, como RS-232, interface de comunicação síncrona, como SPI e I2C, USB, placas de rede, entradas e saída digitais ou analógicas e outros.

2.2 Protocolos de Comunicação

No âmbito de telecomunicações, um protocolo de comunicação é um conjunto de regras que permite que dois ou mais dispositivos de um sistema de comunicação transmitam informação por meio de uma variação de uma quantidade física. Essas são as regras ou padrões que definem a sintaxe, a semântica e a sincronização da comunicação além de possíveis métodos de recuperação de erros. Os protocolos podem ser implementados em *hardware*, *software* ou ambos [13].

Sistemas de comunicação usam formatos(protocolos) bem definidos para a troca de mensagens. Esses protocolos devem ser acordados previamente pelas partes envolvidas. Para alcançar esse acordo, os protocolos podem se tornar padrões técnicos.

Os protocolos devem especificar as regras de uma transmissão de mensagens entre sistemas de comunicação e por isso os seguintes tópicos devem ser definidos: formato de dados, formato de endereço, mapeamento de endereço, roteamento, detecção de erros, perda de informação, controle de sequência, direcão do fluxo de dados e fluxo de dados.

2.2.1 Conjunto de Protocolos de Internet

O conjunto de protocolos de internet é o modelo de conexões e conjunto de protocolos utilizados para a internet e redes semelhantes. Geralmente conhecido como TCP/IP, porque os o protocolo de controle de transmissão, TCP, e o protocolo de internet, IP, foram os primeiros a serem definidos durante o desenvolvimento [14].

O TCP/IP proporciona comunicação ponta a ponta especificando como os dados devem sem empacotados, endereçados, transmitidos, roteados e recebidos. Essa funcionalidade é organizada em quatro camadas abstratas que são utilizadas para classificar os outros protocolos. De baixo para cima, tem-se a camada de ligação, contendo os métodos de comunicação para

dados que se mantenham dentro de um segmento da rede, a camada de internet, conectando as redes independentes, a camada de transporte, que gerencia a comunicação entre *hosts* e a camada de aplicação, que proporciona troca de dados entre processos para as aplicaçãoes.

Protocolo de Transferência de Hipertexto - HTTP

O protocolo de transferência de hipertexto, ou HTTP, é um protocolo de aplicação para sistemas de informação de hipermídia colaborativos e distribuídos [15]. O HTTP é a base da comunicação de dados da World Wide Web.

Hipertexto é um texto estruturado que usa ligações lógicas (hyperlinks) entre nós contendo textp. O HTTP é o protocolo que troca ou transfere hipertexto. Ele funciona como um protocolo de pedido-resposta no modelo cliente-servidor de computação. O cliente submete uma mensagem de pedido HTTP para um servidor. O servidor que providencia os recursos, como arquivos HTML e outros, retorna uma mensagem resposta para o cliente. A respostas contém a informação do estado do pedido e também pode retornar um conteúdo no corpo da mensagem.

Em sua definição, é presumido o uso de um protocolo da camada de transporte confiável e o TCP é geralmente utilizado. Entretanto, o HTTP pode ser adaptado para o uso de protocolos não tão confiáveis como o *User Datagram Protocol*, ou UDP.

Constrained Application Protocol - CoAP

O *constrained application protocol*, ou CoAP, é um protocolo de software criado para ser utilizado em dispositivos eletrônicos bem simples, permitindo a eles comunicação interativa pela internet [16]. É direcionado a pequenos sensore de baixa potência, válvulas e outros que precisam ser controlados ou supervisionados remotamente.

O CoAP é um protocolo da camada de aplicação direcionado a dispositivos com poucos recursos. Foi desenhado para ser de fácil tradução para o HTTP para uma simples integração com a *web* além de possuir recursos especializados como suporte a *multicast*, baixo gasto com processamento em excesso e alta eficiência. E esses são fatores determinantes para configurá-lo de extrema importância para a IoT e dispositivos com comunicação máquina a máquina.

O CoAP usa dois tipos de mensagens, pedidos e respostas, usando um cabeçalho simples de formato binário. Ele é por padrãoligado ao UDP mas pode alternativamente fazer uso de outro protocolo para uma maior segurança. Além disso, oferece suporte para diversas

linguagens como Java, Python, C, Ruby e várias outras.

Message Queue Telemetry Transport - MQTT

O *message queue telemetry transport*, ou MQTT, é um protocolo de mensagens leves para sensores e pequenos dispositivos móveis otimizado para redes TCP/IP não confiáveis e de alta latência, que necessitem de rastro de código pequeno ou possuem banda limitada [17]. O sistema de troca de mensagens é fundamentado no modelo publicador-subscritor e requer um intermediário. Esse intermediário é o responsável por distribuir as mensagens aos clientes interessados baseado no tópico da mensagem.

O MQTT define métodos para indicar a ação desejada a ser realizada. Esses métodos são: Connect, espera uma conexão ser estabelecida com o servidor, Disconnect, espera que o cliente termine todo o trablaho e depois finaliza a sessão, Subscribe, se cadastra para receber atualizações dos valores dos tópicos desejados e Publish, publica um valor para o tópico especificado.

2.3 Yocto Project

O Yocto Project é um projeto colaborativo de código aberto que providencia modelos, ferramentas e métodos para auxiliar a criação de sistemas embarcados baseados em Linux não importando a arquitetura do hardware. O projeto tem como objetivo facilitar o trabalho dos desenvolvedores de sistemas customizados de Linux suportando as arquiteturas ARM, MIPS, PowerPC e x86. Uma parte essencial disso é o sistema de construção, baseado na arquitetura *OpenEmbedded*, que possibilita aos desenvolvedores criarem suas próprias distribuições de Linux especificas para o seu ambiente de trabalho. Essa implementação da *OpenEmbedded* é chamada de Poky [18].

Construir a correta distribuição de Linux para os dispositivos conectados pode ser devagar e cara, além de que juntar todas as peças necessárias para cada dispositivo causa aos desenvolvedores problemas de escalabilidade e incompatibilidade. O Yocto Project consegue minimizar esses problemas pois oferece customização das características de uma distribuição por meio do uso de uma arquitetura com camadas com pacotes pré-moldados, chamados de receitas, que fornecem as instruções de construção para os pacotes de rede, gráficos e muitos outros. É por isso que o Yocto Project é considerado fundamental para o avanço e crescimento da *Internet of Things*.

Um dos diferenciais do Yocto Project é a grande rede de parceiros de hardware que fornecem camadas atualizadas e bem gerenciadas oferecendo suporte as principais arquiteturas e isso fornece uma base para dispositivos bem diferentes poderem ter acesso ao mesmo software e bibliotecas. Outro diferencial é a comunicação pois são necessários diversos padrões de comunicação e protocolos para que os dispositivos possam se comunicar e o Yocto Project oferece camadas que são capazes de realizar a transferência de dados utilizando as diferentes especificações de cada protocolo.

2.4 Programação para Node.js (JavaScript)

JavaScript é uma linguagem de programação interpretada de alto nível que segue as especificações do padrão ECMAScript. Juntamente com HTML e CSS, é uma das três tecnologias fundamentais da produção de conteúdo para internet e a maioria dos sites a emprega e é suportada por todos os navegadores modernos sem a necessidade de uso de plug-ins [19]. Apesar de compartilhar a nomenclatura, a sintaxe e algumas bibliotecas padrão, as linguagens JavaScript e Java não são relacionadas e possuem diferentes semânticas. A sintaxe da JavaScript é derivada da linguagem C.

A linguagem JavaScript possui três principais interpretadores sendo eles: SpiderMonkey, Chakra e, o que vai ser apresentado a seguir, o V8. O V8 é o interpretador de código aberto e alta performance da Google. Ele compila e executa os códigos fonte em JavaScript, gerencia a alocação de memória para objetos e coleta os objetos que não são mais necessários.

JavaScript é comumente utilizado para a programação da parte do cliente em um navegador, sendo utilizado para manipular os objetos DOM, sendo que esses são fornecidos pelo navegador enquanto o V8 fornece as funções, tipos de dados e operadores. E esse interpretador é a engrenagem por trás da ferramenta Node.js que é utilizada para o desenvolvimento da parte do servidor de aplicações Web.

O Node.js é um ambiente de execução multi-plataforma de código aberto para o desenvolvimento dos servidores para aplicações na web. Apesar de não ser estruturado em JavaScript, muitos dos pacotes básicos do Node.js são escritos em JavaScript além dos novos módulos que podem ser criados utilizando essa linguagem. A sua arquitetura é baseada em eventos e é capaz de controlar entradas e saídas assíncronas. Essas escolhas de construção têm como objetivo otimizar o fluxo de dados e a escalabilidade de aplicações Web que necessitam de muitas operações de entrada e saída ou aplicações em tempo real [20].

O Node.js permite a criação de servidores Web e ferramentas de rede utilizando JavaScript e os módulos que estão a disposição para download e fornecem diversas novas funcionalidades, como o Socket.io que é explorado a seguir.

O Socket.io é uma biblioteca de rede para JavaScript que disponibiliza comunicação bidirecional, instantânea e baseada em eventos entre servidor e cliente. Ela utiliza o protocolo WebSocket porém, oferece mais recursos como *broadcast* e *multicast*, armazenamento de dados associados a cada cliente e entrada e saída assíncronas. Essa biblioteca oferece a habilidade de implementar analise de padrões em dados em tempo real, transmissão de imagens, áudio e vídeo, mensagens instantâneas e colaboração em documentos. Serviços que utilizam essa ferramenta são Microsoft Office, Yammer e Zendesk [21].

Outro pacote do Node.js que tem papel fundamental nesse projeto é a biblioteca MQTT (MQ Telemetry Transport) que implementa o protocolo MQTT e com isso abre um canal de comunicação com o Mosquitto que é um programa intermediário que traduz as mensagens recebidas para o padrão aceito pelo receptor. O MQTT é um protocolo de conectividade máquina a máquina e foi desenvolvido como um assinante e publicador muito leve de mensagens. Ele é útil para conexões com locais remotos pois não necessita de muita banda para o recebimento e transmissão. Porém, ele necessita de um programa que distribui as mensagens para os clientes dependendo do tópico especificado na mensagem [22].

2.5 Linux

O Linux é um sistema operacional criado a partir do modelo de distribuição e desenvolvimento de código aberto. O componente mais importante é o Linux kernel desenvolvido em 1991 por Linus Torvald. O kernel é o programa que constitui o centro do sistema operacional pois possui controle sobre tudo que acontece no sistema, por exemplo, é o primeiro programa a ser inicializado ao se ligar o sistema, gerencia as saídas e entradas requisitadas pelo software e gerencia memória.

O sistema foi inicialmente desenvolvido como um sistema operacional gratuito para computadores pessoais baseados na arquitetura x86 da Intel, porém já foi portado para outras plataformas mais do que qualquer outro sistema operacional. Em sua forma original, ele também é o líder em uso em servidores e em supercomputadores. Além disso, ele também roda em sistemas embarcados onde o sistema é colocado no firmware.

Normalmente, o Linux é oferecido em distribuições que englobam o kernel Linux, biblio-

tecas e ferramentas para diversas operações e um gerenciador de pacotes que tem a função de automatizar o processo de instalação, atualização e desinstalação dos programas para o sistema operacional. As distribuições mais conhecidas são: Debian [23], Ubuntu [24], Fedora [25], etc.

2.5.1 Android

O Android é um sistema operacional para *mobiles* desenvolvido pela Google, baseado no kernel Linux e projetado primeiramente para dispositivos móveis com telas sensíveis ao toque como smartphones e tablets. A interface é baseada em sua maior parte na manipulação direta, usando gestos e toques que correspondem a ações do mundo real, como deslizar e tocar para manipular os objetos na tela juntamente com um teclado virtual para a inserção de texto.

Além dos dispositivos com telas sensíveis ao toque, a Google tem desenvolvido o sistema para televisões, carros, relógios e outras versões do sistema também são utilizadas em notebooks, vídeo games, câmeras digitais e outros aparelhos eletrônicos. Desde 2013, o Android é o sistema operacional mobile mais vendido além de possuir mais de 1,4 bilhão de usuários mensais ativos.

Apesar do código fonte ser liberado pela Google sujeitos as licenças de código livre, o sistema é distribuído juntamente com software proprietário para funcionamento das principais funções oferecidas pela empresa. A natureza de código livre ajudou a criar uma grande comunidade que tenta criar novas funcionalidades ou aprimorar a experiência oferecida pelo sistema.

A empresa oferece um IDE, o Android Studio, para que desenvolvedores possam criar seus próprios aplicativos e também proporciona uma plataforma, a Play Store, para que esses desenvolvedores coloquem seus aplicativos à disposição dos usuários e que também pode gerar receita quando os aplicativos não forem gratuitos.

2.6 Open Internet Consortium

O *Open Internet Consortium* foi fundado pelas empresas que lideram o setor de tecnologia, como Cisco, Intel, General Electric e Samsung, com o intuito de definir os requisitos de conectividade e garantir a interoperabilidade dos bilhões de dispositivos que constituirão a IoT. Isso está sendo realizado por meio de certificações e especificações para oferecer uma

estrutura de conectividade que minimiza a complexidade do processo. Além disso, esse padrão será uma especificação aberta de modo que qualquer pessoa possa implementar e que seja simples para desenvolvedores utilizarem.

2.6.1 Projeto Iotivity

Todas as características citadas acima serão oferecidas para diversas plataformas por meio do projeto *Iotivity* que é patrocinado pelo OIC e visa uma implementação referência com código livre que criará uma arquitetura robusta e extensa que possa ser utilizada para os diversos dispositivos inteligentes e assim facilitar a conectividade entre eles e com a internet [26].

A arquitetura proporcionará uma espécie de mapa a ser seguido pelos fabricantes e provedores de serviço e incluirá:

- Solução comum: definir soluções de comunicação e interoperabilidade entre produtos de múltiplos mercados através de diversos sistemas operacionais e plataformas.

- Protocolos Estabelecidos: criar novos protocolos em comum e utilizar os já existentes para a descoberta e conectividade através de diversos meios.

- Abordagens universais: aplicar abordagens universais para a segurança e identidade.

- Semelhanças definidas: definir perfis comuns, modelos de objetos e interfaces de programação de aplicações (APIs).

- Interoperabilidade: promover interoperabilidade entre dispositivos e aplicações de diversos mercados e de diferentes cenários de uso.

- Oportunidades de inovação: proporcionar oportunidades para inovação e disponibilizar diferenciação.

- Conectividade necessária: conectar tudo desde o menor dispositivo vestível ao maior carro inteligente.

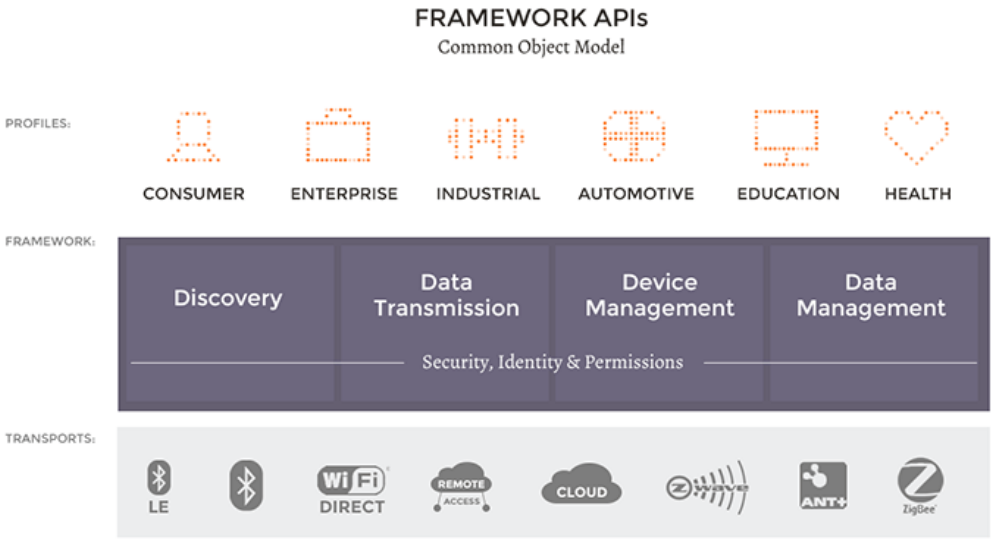

Figura 2.1: Blocos Essenciais da estrutura proporcionada pelo IoTivity [27].

As APIs de estrutura do *IoTivity* proporcionam acesso a estrutura aos desenvolvedores e estão disponíveis em diversas linguagens de programação e para múltiplos sistemas operacionais. A estrutura funciona como um intermediador entre os vários sistemas operacionais e as diversas plataformas de conectividade e possui quatro blocos essenciais:

- Descoberta: Suporte a diversos mecanismos de descoberta para dispositivos e recursos em proximidade e remotamente.
- Transmissão de Dados: Suporte ao controle e a troca de informações baseado em um modelo de mensagens e *streaming*.
- Gestão de Dados: Suporte à coleta, armazenamento e análise de dados de diversas fontes.
- Gestão de Dispositivos: Suporte à configuração, provisionamento e diagnóstico de dispositivos.

Capítulo 3

Materiais e Métodos

Neste capítulo são listados os materiais e métodos utilizados nesse projeto para a criação do sistema proposto representado pela figura 3.1.

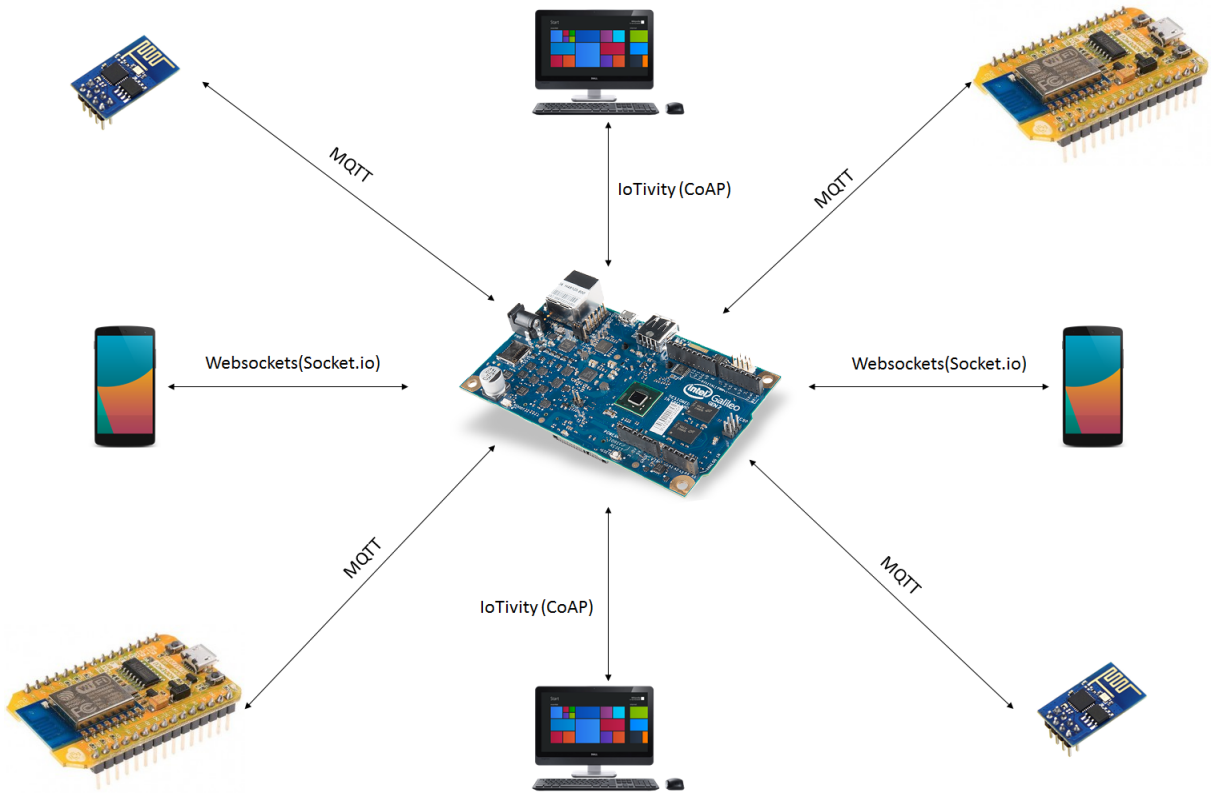

Figura 3.1: Visão geral das possíveis conexões do sistema

3.1 Materiais

Neste projeto foram utilizados os seguintes materiais:

- Computador com sistema operacional Ubuntu
- Intel Galileo
- NodeMCU Devkit v0.9
- Protoboard
- Conectores
- LEDs
- Cartão micro SD
- Adaptador USB Wi-Fi TP-Link WN725N
- Cabo USB para Serial TTL
- Android Studio

A escolha pelo Ubuntu como sistema operacional se deve pelo fato de o projeto IoTivity oferecer compatibilidade com esse sistema e por ele também oferecer todas as ferramentas necessárias para o controle das placas.

A escolha pela Intel Galileo se deu pelo fato de ser a placa que o aluno possuía no momento do desenvolvimento do projeto e que decidiu utilizá-la para testar seus limites frente a aplicação deste trabalho uma vez que a Intel, fabricante da placa, é uma das empresas fundadoras do OIC e do projeto IoTivity.

A escolha da placa NodeMCU DevKit se deu pelo fato de oferecer um ambiente mais completo e integrado quando comparado ao módulo ESP8266, visto que ela possui uma porta USB que serve como porta serial além de oferecer mais entradas e saídas.

O adaptador Wi-Fi foi escolhido para poder oferecer mobilidade e facilidade de instalação da placa Galileo.

A escolha pelo Android Studio para o desenvolvimento do aplicativo foi realizada para testar um novo IDE para a confecção de aplicativos Android uma vez que na disciplina de Projeto de Sistemas Digitais cursada anteriormente foi utilizado o Intel XDK para essa atividade e os resultados não foram satisfatórios.

3.1.1 Intel Galileo

A placa de desenvolvimento Intel Galileo é a primeira placa microcontrolada que possui um processador Intel Quark e hardware e software com compatibilidade de pinos com os shields produzidos para Arduino. Essa plataforma oferece suporte aos sistemas operacionais mais utilizados no mundo além de oferecer compatibilidade ao ambiente de desenvolvimento do Arduino. Na parte de hardware, a placa possui um processador Quark X1000 de 32 bits com clock de 400 MHz, 256 MB de memória RAM e 8 MB de memória flash além de oferecer diversos outros recursos, como por exemplo, 20 entradas e saídas digitais, 6 entradas analógicas, 6 saídas PWM, comunicação serial, porta Ethernet, adaptador para cartão SD, JTAG para debug [9].

Na parte de software, além de oferecer compatibilidade com o ambiente de desenvolvimento do Arduino, o recurso mais importante para este trabalho é a possibilidade de rodar um sistema operacional baseado em Linux especialmente projetado para essa placa utilizando o Yocto Project. Com isso, temos acesso a inúmeras funções presentes no Linux, como SSH, criação de servidores HTTP e também acesso a diversos avanços que estão sendo feitos na área de IoT e que não estão disponíveis para outros sistemas operacionais.

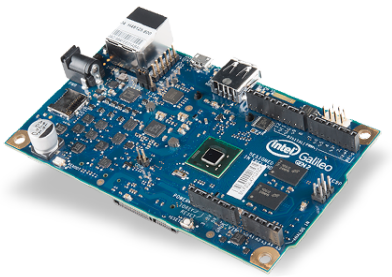

Figura 3.2: Placa Intel Galileo.

3.1.2 ESP8266

O módulo ESP8266 é um SoC, sistema em um chip, com Wi-Fi embutido que possui diversas variantes nas quais cada uma oferece recursos diferentes como mais portas de entrada e saída, entradas analógicas entre outros. Esse módulo é capaz de hospedar uma aplicação ou se tornar responsável por todas as funções de rede de outro processador, como por exemplo substituir o shield ethernet para um Arduino [11].

O módulo possui um processador de 32 bits com arquitetura RISC produzido pela Tensilica operando a 80 MHz, 64 KB de memória ROM para o boot além de 64 KB de RAM para

instruções e 96 KB de RAM para dados. O módulo oferece suporte ao padrão IEEE 802.11 b/g/n de Wi-Fi e dependendo do modelo pode oferecer até 16 pinos de entrada e saída.

Ele também oferece uma grande possibilidade de desenvolvimento de aplicações na área de IoT uma vez que ele possui poder de processamento e espaço de armazenamento suficientes e que pode ser facilmente integrado a diversos sensores.

Devido as funcionalidades oferecidas, foram criadas diversas placas de desenvolvimento que utilizam o ESP8266 como seu processador central e adicionam outros recursos a ele. No caso desse trabalho, será utilizada uma placa de desenvolvimento chamada NodeMCU DevKit v0.9 que acrescenta ao módulo em sua variante ESP-12, uma porta USB-TTL Serial que permite a programação do módulo utilizando somente um cabo USB, além de deixá-lo compatível com o layout da protoboard.

Essa placa oferece diferentes métodos de programação, sendo eles utilizando o NodeMCU e a linguagem LUA, utilizando o Arduino IDE e a linguagem C ou via comandos AT.

Figura 3.3: Módulo ESP8266 e NodeMCU DevKit v0.9.

3.2 Métodos

O primeiro passo para realizar esse projeto foi a preparação do computador rodando o Ubuntu para que ele pudesse ser capaz de compilar a imagem personalizada baseada no Linux utilizando o Yocto Project. Utilizando o gerenciador de pacotes do Ubuntu, apt-get, instalou-se os seguintes pacotes: debootstrap, build-essential, git, p7zip-full, chrpath, diffstat, gawk, wget, git-core, diffstat, unzip, texinfo, gcc-multilib, socat, libsdl1.2-dev, xterm e parted. A maioria desses pacotes somente foi utilizada para a compilação da imagem.

Após essas instalações, fez-se necessário obter a versão *daisy* do *poky*, que é a distribuição base do projeto Yocto, e também as diversas outras camadas que personalizarão a imagem fazendo-a assim compatível com a placa Intel Galileo que foi utilizada. As seguintes camadas foram necessárias para habilitar todas as funcionalidades da placa: meta-intel-quark, meta-intel-iot-middleware, meta-intel-galileo, meta-openembedded e meta-intel-iot-devkit. A última camada que foi adicionada ofereceu as funcionalidades do projeto *IoTivity* e pôde ser encontrada pela identificação meta-oic. Todas essas camadas podem ser obtidas via comandos git a partir do endereço git.yoctoproject.org [28].

Após o *download* de todas as camadas fez-se necessário carregar o ambiente de trabalho utilizando o comando source seguido do argumento oe-init-build-env. Em seguida, foi necessário editar o arquivo bblayers.conf contido na pasta conf e adicionar os caminhos para as camadas que foram utilizadas. No mesmo diretório, foi preciso alterar a versão da distribuição no campo DISTRO para 'iot-devkit-multilibc', alterar a classes dos pacotes no campo 'PACKAGE-CLASSES' para 'package-ipk' e alterar a dispositivo alvo no campo "MACHINE" para 'quark' no arquivo 'auto.conf'. Finalmente, para obter-se as funcionalidades proporcionadas pelo projeto IoTivity, foi necessário adicionar as seguintes linhas no mesmo arquivo:

IMAGE-INSTALL-append += "iotivity-resource-samples iotivity-service-samples"

IMAGE-INSTALL-append = "iotivity-dev"

Após terminar a edição do arquivo auto.conf foi iniciada a compilação da imagem. Para começar, foi necessário executar o seguinte comando: bitbake iot-devkit-image. O processo pode levar algumas horas para finalizar e é indispensável ter uma conexão com a internet visto que o processo requer o *download* de alguns arquivos.

Terminado o processo de compilação da imagem, foi utilizado um aplicativo, que acompanha a camada meta-intel-iot-devkit, que criou a imagem final que foi colocada no cartão micro SD e utilizada pela placa. O comando que foi utilizado é o seguinte:
"/meta-intel-iot-devkit/scripts/wic_monkey create -e iot-devkit-image
/meta-intel-iot-devkit/scripts/lib/image/canned-wks/iot-devkit.wks". Após o termino do processo, um caminho foi mostrado na tela e indicou o diretório onde a imagem a ser colocada no cartão SD se encontrava.

Vale ressaltar que alguns exemplos e todos os recursos do projeto *IoTivity* já estão embutidos na imagem criada uma vez que a camada meta-oic foi adicionada ao processo de criação da imagem e estão disponíveis para uso e consulta nos seguintes caminhos: /opt/iotivity e /usr/include/iotivity.

O próximo passo foi inserir o cartão micro SD contendo a imagem na placa, conectar o cabo serial e energizar a placa. Após isso, foi necessário o uso de algum programa que acesse a placa via Serial, nesse caso foi utilizado o aplicativo PuTTy mas existem outras alternativas. Antes de qualquer outro comando, deve-se alterar a data do sistema para que não haja problemas de compatibilidade, por isso usa-se o seguinte comando: 'date mmddHHMMYYYY'. De modo a não necessitar do uso de um cabo Ethernet para ter acesso a internet, deve-se configurar o adaptador USB Wireless. Para isso, mudou-se o diretório para /usr/src/kernel e utilizou-se o seguinte comando: 'make scripts'. Após esse processo terminar, é necessário fazer o *download* do *driver* para o adaptador no computador utilizando o comando 'git clone' seguido desse endereço: 'https://github.com/lwfinger/rtl8188eu'. Depois do download, foi preciso transferir os arquivos obtidos para a placa, mudar o diretório para a pasta criada e executar os comandos em sequência: 'make all' e 'make install'. Após a finalização desse processo, bastou conectar o *dongle* na porta USB da placa, executar o comando 'connmanctl' para iniciar o ambiente de conexão para a wifi. O primeiro passo foi executar o comando 'scan wifi' e em seguida o comando 'services' para obter as redes disponíveis na área e executar 'connect' seguido do *id* da rede desejada. Se a conexão necessitar de uma senha, basta executar o comando 'agent on' antes do comando 'connect' [29].

Após a conexão na rede, fez-se necessário a instalação de alguns pacotes que foram utilizados no código do servidor que é executado via Node.js além de fazer a instalação do Mosquitto que é o programa intermediário que gerencia as mensagens no formato do padrão MQTT. Para a instalação dos pacotes, primeiro deve-se atualizar o 'npm' que é a ferramenta de controle e instalação dos pacotes do Node.js com o comando 'npm update', após isso deve-se executar o seguinte comando para a instalação dos pacotes: 'npm install -g mqtt socket.io', esse comando instala globalmente, ou seja, qualquer aplicação pode utilizá-los, os pacotes MQTT e Socket.io que já foram discutidos anteriormente.

A instalação do Mosquitto difere da instalação dos pacotes, primeiro deve-ser executar o seguinte comando para fazer o download do arquivo compactado: 'wget http://mosquitto.org/files/source/mosquitto_1.4.8.tar.gz'. Após o download é necessário descompactar o arquivo utilizando o seguinte comando: 'tar xzvf mosquitto_1.4.8.tar.gz'. Deve-se mudar o diretório para a pasta contendo o resultado da descompactação usando o comando 'cd mosquitto_1.4.8' e executar o comando 'make' para realizar a instalação do Mosquitto [30]. Com isso a placa está preparada para executar o código que será descrito mais a frente.

O próximo passo foi preparar a placa de desenvolvimento contendo o ESP8266 para re-

ceber o código que controla sensores/atuadores. Para a configuração da placa foi necessário a instalação do Arduino IDE, após a instalação, deve-se entrar na opção preferências e adicionar a seguinte *URL* no campo de *Additional Boards Manager* e clicar OK: 'http://arduino.esp8266.com/stable/package_esp8266com_index.json'. Após isso, entrar no menu *Tools* e no subitem *boards* selecionar o *Boards Manager*, onde aparecerá um pacote contendo as configurações para a programação dos módulos ESP8266, deve-se instalar esse pacote e novamente no menu *Tools* e no subitem *Boards* agora deve-se selecionar a placa alvo da programação, nesse caso a NodeMCU 0.9. Com isso, concluiu-se a configuração do IDE e o próximo passo foi inserir o código e compilá-lo para ser mandado para a placa.

É necessário também preparar o computador no qual será executado os exemplos que foram criados ao se gerar a imagem contendo os arquivos do projeto IoTivity. Após a geração da imagem, também é possível criar um SDK específico para que possa ser realizado o desenvolvimento de novas aplicações para a placa, porém existe um modo mais prático para compilar os novos programas escritos para a placa e para os outros dispositivos que se integrarão ao ambiente proporcionado pelo IoTivity. Basta baixar a última versão disponível para download no site 'iotivity.org' e utilizar a ferramenta *Scons* para compilar os programas feitos para o ambiente IoTivity tanto para os sistemas baseados em Linux quanto para Arduino, Android e Tizen, que é um sistema operacional baseado no kernel do Linux e muito utilizado em aplicações da Samsung [31].

O procedimento para a criação de novos programas não é muito simples pois além de criar um código que esteja em conformidade com os padrões estabelecidos para uma total integração no projeto IoTivity, também se fez necessário editar um arquivo chamado SConscript especificando o nome do novo programa, o seu arquivo fonte e adicioná-lo a uma lista onde ele será importado no ambiente de construção para ser compilado e especificar todos os caminhos para as bibliotecas necessárias. Sendo uma ferramenta de compilação cruzada, o SCons é capaz de criar programas compatíveis com diversos sistemas, como Linux, Arduino e Android, além de oferecer opções para a escolha do transporte de informações, por exemplo, via IP ou via Bluetooth, e também a possibilidade da compilação para diferentes arquiteturas [26].

O último passo antes da criação do códigos foi preparar o computador que recebeu o software Android Studio no qual foi confeccionado o aplicativo para o controle do sistema. O software pode ser encontrado no site para desenvolvedores Android e sua instalação é muito simples visto que ao selecionar o SDK compatível com a última versão do sistema

operacional, todas as funcionalidades das versões anteriores estão presentes. O software oferece um editor de layout com o recurso de arrastar e soltar para facilitar a criação do aplicativo. Para finalizar a preparação do software foi necessário editar o arquivo 'build.gradle' que estava na árvore de arquivos embaixo do item *Gradle Scripts* e adicionar a seguinte linha no campo 'dependencies': 'compile ('io.socket:socket.io-client:0.7.0')'. Com essa linha, o aplicativo é capaz de usar as propriedades oferecidas pela biblioteca Socket.io que fornece um meio mais simples e rápido de comunicação via sockets [21].

Após a preparação das plataformas, o próximo passo foi a confecção e implantação dos códigos para cada dispositivo que serão discutidos a seguir.

3.2.1 Implementação Servidor Galileo

O código do servidor da galileo pode ser dividido em duas partes: a primeira é responsável pela comunicação com o aplicativo e a segunda é responsável pela comunicação com os outros dispositivos. A comunicação com o aplicativo é feita com o uso da biblioteca Socket.io e o modo como é feita é pela criação de um servidor http e adição dos recursos dessa biblioteca ao servidor. Após isso, o servidor fica esperando uma conexão e ao receber essa conexão, uma função é chamada para a recepção e emissão de mensagens para o aplicativo. Por exemplo, ao apertar o botão do aplicativo, uma mensagem específica será enviada para o servidor que estará aguardando essa mensagem para executar outra função.

A segunda parte realiza a conexão, utilizando a biblioteca MQTT, com o servidor Mosquitto rodando na placa e é responsável pela publicação e assinatura das mensagens no protocolo MQTT. Ou seja, essa parte é responsável por enviar e receber as mensagens no formato do protocolo para os outros dispositivos ligados ao sistema.

O fluxograma abaixo ilustra o funcionamento do servidor implementado.

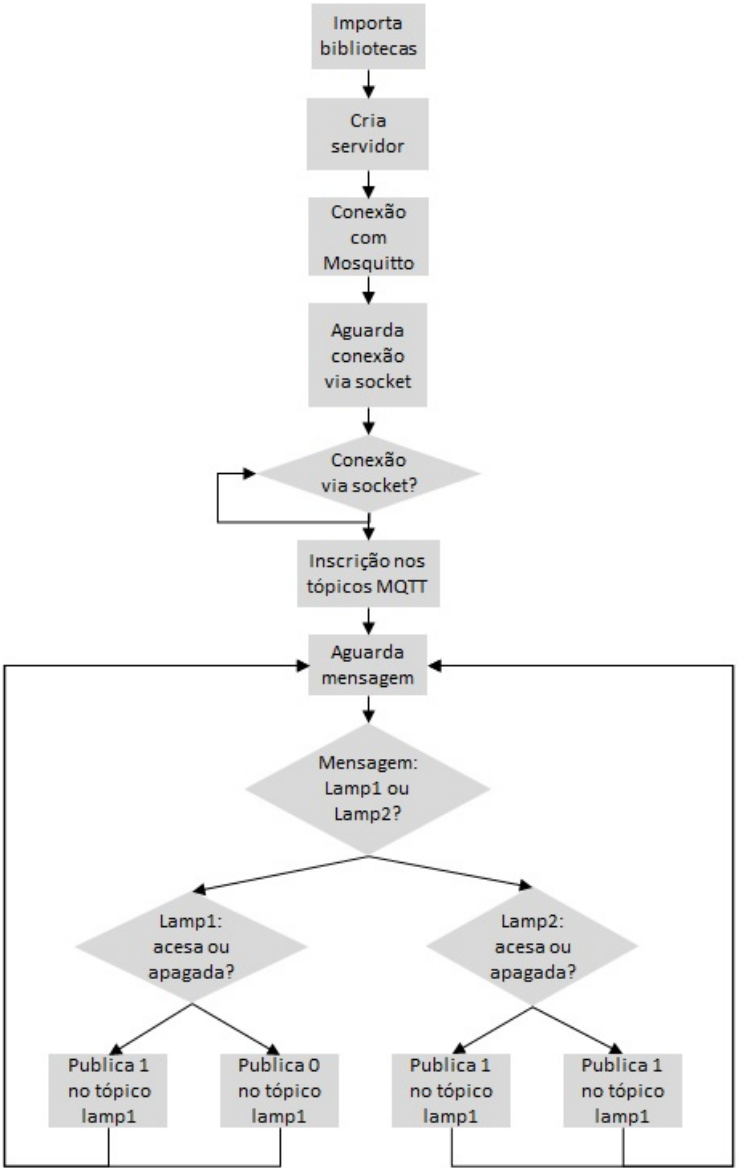

Figura 3.4: Fluxograma Servidor Galileo

3.2.2 Implementação Módulo ESP8266

O código da placa de desenvolvimento que contém o módulo ESP8266 também é dividido em duas partes: a configuração da conexão à rede local sem fio e a assinatura e transmissão de mensagens no protocolo MQTT.

Para a primeira parte é utilizada uma biblioteca especificamente criada para o módulo ESP8266, onde é definido o nome da rede e a senha no código e existe uma tentativa de conexão. Todo esse processo pode ser observado por um monitorador serial pois foram programadas mensagens a serem enviadas via serial para observar o andamento dessa atividade.

Para a segunda parte também é utilizada uma biblioteca especialmente criada para lidar

com as especificações do protocolo MQTT. É definido no código o endereço do servidor Mosquitto, nesse caso o IP da placa Galileo, e uma conexão é estabelecida e a seguir, o programa fica esperando uma mensagem vinda da placa galileo para ativar ou desativar um atuador. Após realizar a ação, uma mensagem é enviada para a placa galileo a fim de confirmar a atividade realizada.

O fluxograma abaixo ilustra o funcionamento do módulo ESP8266.

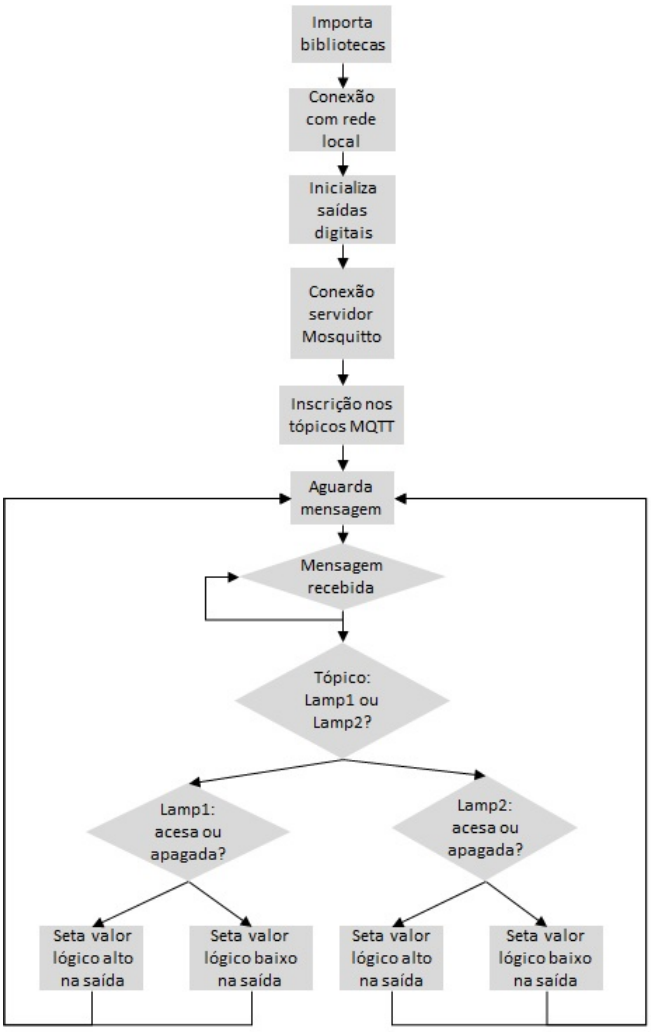

Figura 3.5: Fluxograma ESP8266

3.2.3 Implementação Android

O código para o aplicativo android é o menos complexo pois somente é criada uma função para a conexão com a placa Galileo, novamente utilizando a biblioteca Socket.io, e após isso, são criadas funções de escuta, ou seja, elas ficam esperando a ação do seu determinado botão para enviarem uma mensagem para a placa. Outro detalhe, para cada ação realizada no botão,

um mensagem pop-up aparecerá indicando a ação.

A parte gráfica do aplicativo foi desenvolvida com ajuda da ferramenta arraste e solte do Android Studio. O design escolhido é básico contando com uma tela de login que precede a página principal onde os dispositivos a serem controlados são mostrados em forma de cartão contendo seu nome e o botão para realizar a ação.

3.2.4 Implementação IoTivity

O código utilizado para implementar o projeto IoTivity foi baseado nos exemplos que são obtidos ao criar a imagem do Linux com a camada OIC. O primeiro par de códigos tem como função encontrar o dispositivo conectado e informá-lo sobre as informações da plataforma a qual ele se conectou. No servidor, que está rodando na placa Galileo, são informadas as características da plataforma, importadas as funcionalidades do IoTivity e o servidor emite a informação de que está esperando uma conexão e ao receber uma conexão, as características são enviadas para o cliente. No código do cliente, que foi implementado no dispositivo, uma conexão foi tentada e ao realizá-la, as informações da plataforma foram recebidas.

O segundo par de códigos teve a implantação invertida, ou seja, o servidor está no dispositivo e o cliente está na placa Galileo. Os dois códigos seguem o mesmo padrão de conexão dos outros, onde um recurso disponibilizado pelo IoTivity permite que os dispositivos se declarem disponíveis como ocorre ao se ligar o Bluetooth de um aparelho. O código do servidor simula uma lâmpada que cuja potência subirá em 10 unidades a cada vez que o cliente se conectar para fazer uma observação do recurso. O código do cliente, ao se conectar, busca as informações do servidor e observa os recursos iniciados no servidor e assim receberá as informações sobre a lâmpada.

Capítulo 4

Resultados e Discussões

Os resultados atingiram totalmente o que foi projetado e era esperado. Todas as etapas propostas foram concluídas e apresentaram os resultados esperados.

A confecção da imagem personalizada do Linux baseado do Yocto Project funcionou perfeitamente apesar de terem sido necessárias várias tentativas até que o processo fosse aperfeiçoado e que a imagem final contivesse todas as funcionalidades desejadas. O único ponto que deixou espaço para melhora foi o tamanho da imagem final, que teoricamente deveria ter sido reduzido, porém com a adição do projeto IoTivity e a não remoção de algumas funcionalidades que pudessem ser importantes no futuro, teve um aumento de aproximadamente 300 MB quando comparada com a imagem criada sem o projeto IoTivity. Isso pode impactar futuramente no funcionamento do sistema, porém pode ser facilmente contornado com o uso de um cartão de memória de maior capacidade.

A criação do aplicativo para a plataforma Android oferecia uma grande dificuldade visto que a experiência em programação nas linguagens XML e Java, que são as necessárias para o desenvolvimento do aplicativo, era quase nula. No entanto, isso foi contornado com o uso do editor de layout presente no Android Studio que minimizou a programação em XML enquanto que para a parte com programação em Java foi necessário um empenho maior para o entendimento e aprendizagem da linguagem. Devido a esses fatos, o aplicativo ficou com um design básico, porém funcional.

Apesar da dificuldade encontrada para o desenvolvimento profundo do projeto IoTivity devido a complexidade da documentação e dos poucos exemplos que podem ser encontrados, foi possível observar as funcionalidades que essa implementação pode oferecer, como não precisar informar o IP de cada dispositivo e a conexão ser feita automaticamente, por meio dos exemplos fornecidos porém, a complexidade do projeto precisa ser diminuída se o intuito

é popularizar essa tecnologia entre os desenvolvedores.

A escolha pelo método de compilação pela ferramenta SCons, ao invés do uso do SDK criado no processo de construção da imagem personalizada, provou-se somente adequado visto que uma das funcionalidades esperadas era a possibilidade de compilação dos códigos direto na placa, porém devido à falta de memória RAM e pouco poder de processamento esse processo se mostrou impossível de ser realizado. Outro fato que pesou contra o uso do SCons foi a necessidade de alterar um arquivo que contém todos os códigos a serem compilados, apesar de ser uma tarefa simples, até a descoberta de que esse processo, que não estava especificado na documentação fornecida, era necessário foi gasto muito tempo que poderia ter sido dedicado a outra atividade.

Foram realizados dois testes, um para obter o desempenho do sistema e outro, para confirmar a presença das novas funcionalidades acrescentadas pelo projeto *IoTivity*. Vale ressaltar previamente que esses testes foram realizados em um ambiente que possuía boa cobertura de sinal de WiFi além de todos os dispositivos estarem conectados à mesma rede local.

O primeiro teste realizado foi o de latência, que mediu a demora de comunicação entre os dispositivos. Foram realizadas as medidas dos atrasos da conexão e do envio de uma mensagem a partir do aplicativo Android para a placa Galileo utilizando um função no código que marcava o início da ação e ao final dela mostrava na tela quanto tempo havia se passado desde a maração inicial. Os resultados são apresentados na tabela 4.1 e nas figuras 4.1 e 4.2 .

```
root@galileo:~/projects# node server_galileo.js
Server Online!
user connected
Tempo gasto na conexao: 3ms
```

Figura 4.1: Atraso da conexão

```
lamp1 on!
Tempo gasto na mensagem: 3ms
```

Figura 4.2: Atraso da mensagem

Tipo de Atraso	Tempo de atraso médio (ms)
Conexão	3
Mensagem Websocket	3
Total	**6**

Tabela 4.1: Atrasos do sistema

Observando esses valores podemos ver que o atraso acumulado é pequeno quando consideramos que o mínimo de atraso que o olho humano pode perceber, segundo diversos estudos, é de 14 ms. Por causa disso, tem-se a impressão de que a ação ocorre instantaneamente.

O outro teste realizado teve como objetivo comprovar que a placa Galileo possuía as novas funcionalidades adicionadas pelo projeto *IoTivity*. Para isso, foram inicializados os códigos na Galileo e no computador com a distribuição Ubuntu do Linux e observado as informações que apareciam na sessão da placa Galileo. As figuras 4.3 e 4.4 servem como evidência para comprovar que a placa Galileo está preparada para executar exemplos que utilizam os recursos do projeto *IoTivity*.

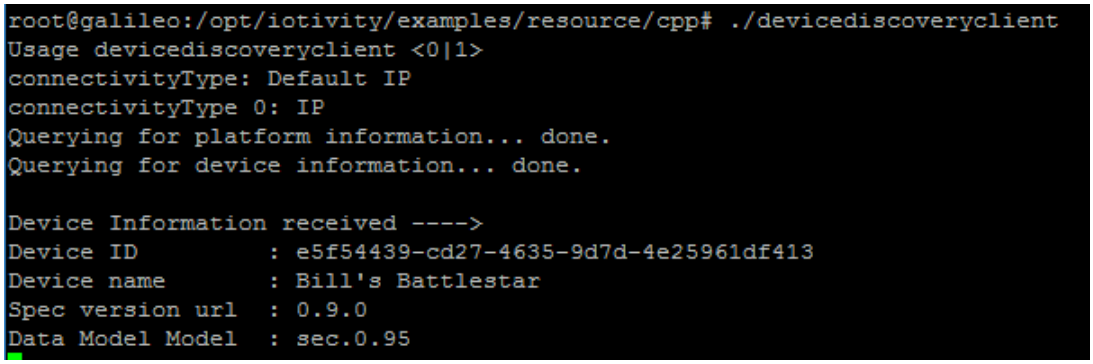

Figura 4.3: Atraso da conexão

```
root@galileo:/opt/iotivity/examples/resource/cpp# clear
root@galileo:/opt/iotivity/examples/resource/cpp# ./simpleserver

Usage : simpleserver <value>
    Default - Non-secure resource and notify all observers
    1 - Non-secure resource and notify list of observers

    2 - Secure resource and notify all observers
    3 - Secure resource and notify list of observers

    4 - Non-secure resource, GET slow response, notify all observers
Created resource.
Added Interface and Type
Waiting
0:
In entity handler wrapper:

        In Server CPP entity handler:
                requestFlag : Request
                        requestType : GET
0:
In entity handler wrapper:

        In Server CPP entity handler:
                requestFlag : Request
                        requestType : PUT
                                state: 1
                                power: 15
0:
In entity handler wrapper:

        In Server CPP entity handler:
                requestFlag : Request
                        requestType : POST
0:
In entity handler wrapper:

        In Server CPP entity handler:
                requestFlag : Request
                        requestType : POST
                                state: 1
                                power: 55
0:
In entity handler wrapper:

        In Server CPP entity handler:
                requestFlag : Request
                        requestType : GET
                requestFlag : Observer

Power updated to : 65
```

Figura 4.4: Atraso da mensagem

Capítulo 5

Conclusões

Considerando o objetivo definido no início do projeto, o trabalho realizado e os resultados obtidos, pode-se concluir que esse projeto além de conseguir deixar um protótipo funcional e preparado para novas funcionalidades também deixou uma sólida base para futuros trabalhos utilizando essa tecnologia.

O ponto mais importante do trabalho foi o sucesso na integração do projeto IoTivity à placa de desenvolvimento escolhida e também o fato de conseguir produzir um protótipo que além de oferecer compatibilidade com o IoTivity, também é capaz de gerenciar, de maneira ágil, diversos dispositivos e sensores que podem ser incorporados futuramente ao projeto.

Alguns pontos no qual o trabalho produziu o resultado projetado são a criação da imagem personalizada que já continha a estrutura, as funcionalidades e os padrões definidos pelo Open Internet Consortium, uma vez que ofereceu um recurso muito importante para futuros projetos que seria maximizar a eficiência e minimizar o espaço gasto ao se retirar funcionalidades não desejadas, a integração de diferentes plataformas, variando desde o módulo ESP8266 até o Android, e a criação de um sistema que oferece um tempo de resposta que cria a impressão de ser instantâneo para o usuário.

Um dos pontos mais importante desse trabalho foi agregar novos conhecimento ao aluno visto que apesar de possuir experiência prévia com as linguagens JavaScript e C e a placa Galileo, foi necessário desenvolver novas habilidade para a programação do aplicativo Android que foi realizada em XML e Java, duas linguagens onde o conhecimento prévio era quase inexistente. Além do aprendizado de novas linguagens, o projeto também proporcionou a oportunidade de explorar uma nova plataforma que possui uma grande gama de aplicações para a internet das coisas e que possui uma otina perspectiva de futuro quando considerado os avanços recentes na área de hardware.

Apesar de não ter sido possível alcançar um nível de desenvolvimento superior ao proposto e projetado, a experiência com o projeto IoTivity foi de grande valia uma vez que os exemplos fornecidos funcionaram perfeitamente além de que a indústria deve seguir os requisitos e padrões estabelecidos pelo OIC criando assim dispositivos compatíveis com os da concorrência para que se possa gerar em ambiente onde não seja preciso utilizar diversos aplicativos ou adaptadores e tudo poderá ser controlado por um único aplicativo. As causas para o fato de não superar o que foi proposto foram a complexidade apresentada pelo projeto, o pouco desenvolvimento tanto para a Galileo quanto para outros sistemas e a documentação imprecisa ou confusa. Um bom exemplo dessa situação é que a cada atualização do projeto era necessário gerar novamente a imagem personalizada e realizar todo o processo de configuração da placa Galileo novamente, além disso mesmo se a atualização não fosse realizada para não despender tempo com essa atividade, ainda havia o risco de que alguma ferramenta tenha sido descontinuada ou uma nova seja adicionada deixando assim a placa defasada em relação aos avanços.

Algumas dificuldades já eram esperadas visto que é o início do projeto, poucas plataformas são suportadas, há muitas especificações a serem seguidas e acima de tudo, o desenvolvimento está sendo feito em conjunto com a comunidade, o que por um lado é extremamente importante já que as pessoas envolvidas diretamente com o uso dessa tecnologia podem sugerir mudanças, no entanto por outro lado, se não houver uma grande adoção por parte da comunidade, o projeto pode se tornar obsoleto ou conter funcionalidades que não serão utilizadas.

Todos esses problemas devem ser resolvidos quando houver um consenso de que esse será o novo padrão a ser seguido pela indústria e o primeiro passo para isso já foi dado com o anúncio da Open Connectivity Foundation, OCF, que integrou Microsoft, Qualcomm e Electrolux ao integrantes do OIC. A OCF possui o mesmo objetivo do OIC que é definir especificações e certificações com o intuito de oferecer interoperabilidade confiável, ou seja, uma plataforma de conectividade que reduz a complexidade porém, engloba mais parceiros com grande representatividade tanto no mercado de dispositivos móveis quanto no cenário geral de tecnologia, o que pode ser um grande diferencial quando comparado ao OIC.

5.1 Sequência do Trabalho

Os próximos passos do projeto consistem em integrar novos dispositivos ao sistema, novas funcionalidades, testar diferentes placas de desenvolvimento visto que a fabricante da Galileo não está mais tão dedicada a seu desenvolvimento além de existirem alternativas melhores e mais baratas e o que seria o cenário considerado ideal para o futuro, a extinção da central de comando, fazendo com que os dispositivos comunicassem entre si e diretamente com o aplicativo criado para smartphones.

Além disso, a aplicação do projeto IoTivity em outras plataformas é um passo seguinte, visto que o projeto oferece compatibilidade com outras plataformas como Arduino e Android e oferece diversas funcionalidades que serão de extrema importância no desenvolvimento futuro da Internet das Coisas.

Referências Bibliográficas

[1] History of the internet of things. http://http://postscapes.com/internet-of-things-history, Acesso em: 09 de outubro de 2015.

[2] Gartner's 2011 hype cycle special report evaluates the maturity of 1,900 technologies. http://www.gartner.com/newsroom/id/1763814, Acesso em: 13 de novembro de 2015.

[3] Gartner's 2014 hype cycle for emerging technologies maps the journey to digital business. http://www.gartner.com/newsroom/id/2819918, Acesso em: 14 de novembro de 2015.

[4] Neil Gross. The earth will don an electronic skin. http://www.businessweek.com/1999/99_35/b3644024.htm, Acesso em: 14 de novembro de 2015.

[5] S. Heath. Embedded systems design, 2002.

[6] Raspberry pi. https://www.raspberrypi.org/, Acesso em: 20 de maio de 2016.

[7] Beaglebone black. https://beagleboard.org/black, Acesso em: 20 de maio de 2016.

[8] Arduino. https://www.arduino.cc/, Acesso em: 20 de maio de 2016.

[9] Placa de desenvolvimento intel galileo. http://www.intel.com.br/content/www/br/pt/do-it-yourself/galileo-maker-quark-board.html, Acesso em: 17 de junho de 2015.

[10] Toradex colibri. https://www.toradex.com/pt_br/computer-on-modules/colibri-arm-family, Acesso em: 20 de maio de 2016.

[11] Wifi module - esp8266. https://www.sparkfun.com/products/13678, Acesso em: 10 de setembro de 2015.

[12] Intel edison. https://software.intel.com/pt-br/iot/hardware/edison, Acesso em: 20 de maio de 2016.

[13] L. J. Rodríguez-Aragón. Tema 4: Internet y teleinformática. http://www.uclm.es/profesorado/licesio/Docencia/IB/IBTema4.pdf, Acesso em: 10 de abril de 2016.

[14] Tcp/ip. http://www.itprc.com/tcpipfaq/, Acesso em: 20 de maio de 2016.

[15] Http. https://tools.ietf.org/html/rfc2616, Acesso em: 20 de maio de 2016.

[16] Coap. http://hinrg.cs.jhu.edu/joomla/images/stories/IPSN_2011_koliti.pdf, Acesso em: 20 de maio de 2016.

[17] Mqtt. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html, Acesso em: 20 de maio de 2016.

[18] About | yocto project. https://www.yoctoproject.org/about, Acesso em: 17 de junho de 2015.

[19] D. Flanagan. Javascript: The definitive guide, 2011.

[20] About | node.js. https://nodejs.org/en/about/, Acesso em: 22 de agosto de 2015.

[21] Socket.io. http://socket.io/, Acesso em: 20 de junho de 2015.

[22] Mqtt. http://mqtt.org/, Acesso em: 15 de dezembro de 2015.

[23] Debian. https://www.debian.org/index.pt.html, Acesso em: 20 de maio de 2016.

[24] Ubuntu. http://www.ubuntu.com/, Acesso em: 20 de maio de 2016.

[25] Fedora. https://getfedora.org/pt_BR/, Acesso em: 20 de maio de 2016.

[26] Documentation | iotivity. https://www.iotivity.org/documentation, Acesso em: 05 de julho de 2015.

[27] Architecture overview | iotivity. https://www.iotivity.org/documentation/architecture-overview, Acesso em: 05 de julho de 2015.

[28] Iot- creating a yocto image for the intel galileo using split layers. https://software.intel.com/en-us/blogs/2015/03/04/creating-a-yocto-image-for-the-intel-galileo-board-using-split-layers, Acesso em: 28 de julho de 2015.

[29] Criar imagem yocto para galileo. http://cear.ufpb.br/~isaac/site/tutoriais/intel-galileo/yocto-galileo, Acesso em: 04 de fevereiro de 2016.

[30] Building and running mosquitto on intel edison. https://software.intel.com/en-us/blogs/2015/02/20/building-and-running-mosquitto-mqtt-on-intel-edison, Acesso em: 10 de janeiro de 2016.

[31] About | tizen. https://www.tizen.org/about, Acesso em: 20 de maio de 2016.

Apêndice A

Códigos criados para o projeto

A.1 Código Galileo

```
var http = require('http');
var server = http.createServer();
var io = require('socket.io').listen(server);
var mqtt = require('mqtt');
var clientMQTT = mqtt.connect('mqtt://localhost');
io.on('connection', function(socket){
var start1 = new Date();
console.log('user connected');
ClientMQTT.on('connect',function(){
clientMQTT.subscribe('lamp1');
clientMQTT.subscribe('lamp2');
});
var end1 = new Date() - start1;
console.info("Tempo gasto na conexao: %dms", end1);
socket.on('lamp1',function(state){
if (state==true){
clientMQTT.publish('lamp1',"on");
console.log("lamp1 on!");
var end2 = new Date() - start2;
console.info("Tempo gasto na mensagem: %dms", end2);
} else {clientMQTT.publish('lamp1',"off");
console.log("lamp1 off!");}});
socket.on('lamp2',function(state){
if (state==true){
clientMQTT.publish('lamp2',"on");
} else {clientMQTT.publish('lamp2',"off");
```

```
console.log("lamp2 off!");}
});});
server.listen(3000);
console.log('Server Online!');
```

A.2 Código ESP8266

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Update these with values suitable for your network.

const char* ssid = "TripMate-F32A";
const char* password = "thi301091";
const char* mqtt_server = "10.10.10.27";

WiFiClient espClient;
PubSubClient client(espClient);

char msg[50];
int value = 0;

void setup_wifi() {

delay(10);
// We start by connecting to a WiFi network
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}
```

```
void callback(char* topic, byte* payload, unsigned int length) {
Serial.print("Message arrived [");
Serial.print(topic);
char* topico = topic;
Serial.print("] ");
for (int i = 0; i < length; i++) {
Serial.print((char)payload[i]);
}
Serial.println();

//Switch on the LED if an 1 was received as first character

if((char)topic[4]=='1'){
if ((char)payload[0] == '1') {
Serial.print("lamp 1 acesa");
digitalWrite(12, HIGH);   // Turn the LED on (Note that LOW is the voltage level
// but actually the LED is on; this is because
// it is acive low on the ESP-01)
} else {
Serial.print("lamp1 desligada");
digitalWrite(12, LOW);  // Turn the LED off by making the voltage HIGH
}
} else if((char)topic[4]=='2'){
if ((char)payload[0] == '1') {
Serial.print("lamp2 acesa");
digitalWrite(13, HIGH);   // Turn the LED on (Note that LOW is the voltage level
// but actually the LED is on; this is because
// it is acive low on the ESP-01)
} else {
Serial.print("lamp 2 desligada");
digitalWrite(13, LOW);  // Turn the LED off by making the voltage HIGH
}
}
}

void reconnect() {
// Loop until we're reconnected
while (!client.connected()) {
Serial.print("Attempting MQTT connection...");
// Attempt to connect
if (client.connect("ESP8266Client")) {
```

```
Serial.println("connected");
// Once connected, publish an announcement...
//client.publish("presence", "on");
// ... and resubscribe
client.subscribe("lamp1");
client.subscribe("lamp2");

} else {
Serial.print("failed, rc=");
Serial.print(client.state());
Serial.println(" try again in 5 seconds");
// Wait 5 seconds before retrying
delay(5000);
}
}
}

void setup() {
pinMode(12, OUTPUT);
pinMode(13, OUTPUT);// Initialize the BUILTIN_LED pin as an output
Serial.begin(115200);
setup_wifi();
client.setServer(mqtt_server, 1883);
client.setCallback(callback);
}

void loop() {

if (!client.connected()) {
reconnect();
}
client.loop();



}
```

A.3 Código Android

```
private Socket mSocket;
{try{
mSocket = IO.socket("http://192.168.0.101:3000");
} catch (URISyntaxException e){}}
```

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main_layout);
mSocket.connect();
mSocket.on("connected", onNewMessage);
Switch b1 = (Switch) findViewById(R.id.switch1);
if (b1 != null){
b1.setOnCheckedChangeListener(this);
}
Switch b2 = (Switch) findViewById(R.id.switch2);
if (b2 != null){
b2.setOnCheckedChangeListener(this);
}}
public void connection(){
Log.d("tag","executando a funcao");
//Toast.makeText(MainLayout.this ,"Conectado",Toast.LENGTH_SHORT).show();}
public void onClick (View button){
PopupMenu popup = new PopupMenu(this ,button);
popup.getMenuInflater().inflate(R.menu.pop_menu ,popup.getMenu());
popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener(){
public boolean onMenuItemClick(MenuItem item){
Toast.makeText(MainLayout.this ,"Clicked popup menu item "+ item.getTitle(),
Toast.LENGTH_SHORT).show();
return true;
}});
popup.show();}
@Override
public void onCheckedChanged (CompoundButton buttonView, boolean isChecked){
Toast.makeText(this, "Lampada " + buttonView.getText() +
(isChecked ? " Ligada" : " Desligada"),
Toast.LENGTH_SHORT).show();
mSocket.emit("teste1");}
private Emitter.Listener onNewMessage = new Emitter.Listener(){
@Override
public void call(Object... args){
Log.d("tag","entrou no listener");
//Toast.makeText(MainLayout.this ,"Conectado",Toast.LENGTH_SHORT).show();
connection();}};}
```

Anexo I

Códigos utilizados

I.1 Código IoTivity

I.1.1 Código Placa

```
#include <functional>
#include <pthread.h>
#include <mutex>
#include <condition_variable>
#include "OCPlatform.h"
#include "OCApi.h"
using namespace OC;
using namespace std;
namespace PH = std::placeholders;
int gObservation = 0;
void * ChangeLightRepresentation (void *param);
void * handleSlowResponse (void *param, std::shared_ptr<OCResourceRequest> pRequest);
// Specifies where to notify all observers or list of observers
// false: notifies all observers
// true: notifies list of observers
bool isListOfObservers = false;
// Specifies secure or non-secure
// false: non-secure resource
// true: secure resource
bool isSecure = false;
/// Specifies whether Entity handler is going to do slow response or not
bool isSlowResponse = false;
// Forward declaring the entityHandler
/// This class represents a single resource named 'lightResource'. This resource has
/// two simple properties named 'state' and 'power'
class LightResource{
public:
/// Access this property from a TB client
std::string m_name;
```

```
bool m_state;
int m_power;
std::string m_lightUri;
OCResourceHandle m_resourceHandle;
OCRepresentation m_lightRep;
ObservationIds m_interestedObservers;
public:
/// Constructor
LightResource()
:m_name("Thiago's light"), m_state(false), m_power(0), m_lightUri("/a/light"),
m_resourceHandle(nullptr) {
// Initialize representation
m_lightRep.setUri(m_lightUri);
m_lightRep.setValue("state", m_state);
m_lightRep.setValue("power", m_power);
m_lightRep.setValue("name", m_name);}
/* Note that this does not need to be a member function: for classes you do not have
access to, you can accomplish this with a free function: */
/// This function internally calls registerResource API.
void createResource(){
//URI of the resource
std::string resourceURI = m_lightUri;
//resource type name. In this case, it is light
std::string resourceTypeName = "core.light";
// resource interface.
std::string resourceInterface = DEFAULT_INTERFACE;
// OCResourceProperty is defined ocstack.h
uint8_t resourceProperty;
if(isSecure){
resourceProperty = OC_DISCOVERABLE | OC_OBSERVABLE | OC_SECURE;
}else{
resourceProperty = OC_DISCOVERABLE | OC_OBSERVABLE;}
EntityHandler cb = std::bind(&LightResource::entityHandler, this,PH::_1);
// This will internally create and register the resource.
OCStackResult result = OCPlatform::registerResource(
m_resourceHandle, resourceURI, resourceTypeName,
resourceInterface, cb, resourceProperty);
if (OC_STACK_OK != result){
cout << "Resource creation was unsuccessful\n";}}
OCStackResult createResource1()
{// URI of the resource
std::string resourceURI = "/a/light1";
// resource type name. In this case, it is light
std::string resourceTypeName = "core.light";
// resource interface.
std::string resourceInterface = DEFAULT_INTERFACE;
// OCResourceProperty is defined ocstack.h
uint8_t resourceProperty;
```

```
if(isSecure){
resourceProperty = OC_DISCOVERABLE | OC_OBSERVABLE | OC_SECURE;
}else{
resourceProperty = OC_DISCOVERABLE | OC_OBSERVABLE;}
EntityHandler cb = std::bind(&LightResource::entityHandler, this,PH::_1);
OCResourceHandle resHandle;
// This will internally create and register the resource.
OCStackResult result = OCPlatform::registerResource(
resHandle, resourceURI, resourceTypeName,
resourceInterface, cb, resourceProperty);
if (OC_STACK_OK != result)
{cout << "Resource creation was unsuccessful\n";}
return result;}
OCResourceHandle getHandle()}
return m_resourceHandle;}
// Puts representation.
// Gets values from the representation and
// updates the internal state
void put(OCRepresentation& rep)
{try {
if (rep.getValue("state", m_state))
{cout << "\t\t\t\t" << "state: " << m_state << endl;
}else{
cout << "\t\t\t\t" << "state not found in the representation" << endl;
}if (rep.getValue("power", m_power))
{cout << "\t\t\t\t" << "power: " << m_power << endl;
}else{
cout << "\t\t\t\t" << "power not found in the representation" << endl;
}}catch (exception& e)
{cout << e.what() << endl;}}
// Post representation.
// Post can create new resource or simply act like put.
// Gets values from the representation and
// updates the internal state
OCRepresentation post(OCRepresentation& rep)
{static int first = 1;
// for the first time it tries to create a resource
if(first)
{first = 0;
if(OC_STACK_OK == createResource1())
{OCRepresentation rep1;
rep1.setValue("createduri", std::string("/a/light1"));
return rep1;}}
// from second time onwards it just puts
put(rep); return get();}
// gets the updated representation.
// Updates the representation with latest internal state before
// sending out.
```

```
OCRepresentation get(){
m_lightRep.setValue("state", m_state);
m_lightRep.setValue("power", m_power);
return m_lightRep;}
void addType(const std::string& type) const
{OCStackResult result = OCPlatform::bindTypeToResource(m_resourceHandle, type);
if (OC_STACK_OK != result){
cout << "Binding TypeName to Resource was unsuccessful\n";}}
void addInterface(const std::string& interface) const
{OCStackResult result = OCPlatform::bindInterfaceToResource(m_resourceHandle, interface);
if (OC_STACK_OK != result){
cout << "Binding TypeName to Resource was unsuccessful\n";}}
private:
// This is just a sample implementation of entity handler.
// Entity handler can be implemented in several ways by the manufacturer
OCEntityHandlerResult entityHandler(std::shared_ptr<OCResourceRequest> request)
{cout << "\tIn Server CPP entity handler:\n";
OCEntityHandlerResult ehResult = OC_EH_ERROR;
if(request){
// Get the request type and request flag
std::string requestType = request->getRequestType();
int requestFlag = request->getRequestHandlerFlag();
if(requestFlag & RequestHandlerFlag::RequestFlag)
{cout << "\t\trequestFlag : Request\n";
auto pResponse = std::make_shared<OC::OCResourceResponse>();
pResponse->setRequestHandle(request->getRequestHandle());
pResponse->setResourceHandle(request->getResourceHandle());
// Check for query params (if any)
QueryParamsMap queries = request->getQueryParameters();
if (!queries.empty())
{std::cout << "\nQuery processing upto entityHandler" << std::endl;
}for (auto it : queries){
std::cout << "Query key: " << it.first << " value : " << it.second
<< std:: endl;}
// If the request type is GET
if(requestType == "GET")
{cout << "\t\t\trequestType : GET\n";
if(isSlowResponse) // Slow response case
{static int startedThread = 0;
if(!startedThread){
std::thread t(handleSlowResponse, (void *)this, request);
startedThread = 1; t.detach();}
ehResult = OC_EH_SLOW;}
else // normal response case.{
pResponse->setErrorCode(200);
pResponse->setResponseResult(OC_EH_OK);
pResponse->setResourceRepresentation(get());
if(OC_STACK_OK == OCPlatform::sendResponse(pResponse))
```

```
{ehResult = OC_EH_OK;}}}
else if(requestType == "PUT"){
cout << "\t\t\trequestType : PUT\n";
OCRepresentation rep = request->getResourceRepresentation();
// Do related operations related to PUT request
// Update the lightResource
put(rep);
pResponse->setErrorCode(200);
pResponse->setResponseResult(OC_EH_OK);
pResponse->setResourceRepresentation(get());
if(OC_STACK_OK == OCPlatform::sendResponse(pResponse))
{ehResult = OC_EH_OK;}
else if(requestType == "POST"){
cout << "\t\t\trequestType : POST\n";
OCRepresentation rep = request->getResourceRepresentation();
// Do related operations related to POST request
OCRepresentation rep_post = post(rep);
pResponse->setResourceRepresentation(rep_post);
pResponse->setErrorCode(200);
if(rep_post.hasAttribute("createduri")){
pResponse->setResponseResult(OC_EH_RESOURCE_CREATED);
pResponse->setNewResourceUri(rep_post.getValue<std::string>("createduri"));}
else{
pResponse->setResponseResult(OC_EH_OK);
}
if(OC_STACK_OK == OCPlatform::sendResponse(pResponse))
{ehResult = OC_EH_OK;}}
else if(requestType == "DELETE")
{cout << "Delete request received" << endl;}}
if(requestFlag & RequestHandlerFlag::ObserverFlag)
{ObservationInfo observationInfo = request->getObservationInfo();
if(ObserveAction::ObserveRegister == observationInfo.action)
{m_interestedObservers.push_back(observationInfo.obsId);}
else if(ObserveAction::ObserveUnregister == observationInfo.action)
{m_interestedObservers.erase(std::remove(
m_interestedObservers.begin(),
m_interestedObservers.end(),
observationInfo.obsId),
m_interestedObservers.end());}
pthread_t threadId;
cout << "\t\trequestFlag : Observer\n";
gObservation = 1;
static int startedThread = 0;
// Observation happens on a different thread in ChangeLightRepresentation function.
// If we have not created the thread already, we will create one here.
if(!startedThread)
{pthread_create (&threadId, NULL, ChangeLightRepresentation, (void *)this);
startedThread = 1;}ehResult = OC_EH_OK;}}
```

```
else{std::cout << "Request invalid" << std::endl;
}return ehResult;}};
// ChangeLightRepresentaion is an observation function,
// which notifies any changes to the resource to stack
// via notifyObservers
void * ChangeLightRepresentation (void *param)
{LightResource* lightPtr = (LightResource*) param;
// This function continuously monitors for the changes
while (1){sleep (3);
if (gObservation){
// If under observation if there are any changes to the light resource
// we call notifyObservors
// For demostration we are changing the power value and notifying.
lightPtr->m_power += 10;
cout << "\nPower updated to : " << lightPtr->m_power << endl;
cout << "Notifying observers with resource handle: " << lightPtr->getHandle() << endl;
OCStackResult result = OC_STACK_OK;
if(isListOfObservers){
std::shared_ptr<OCResourceResponse> resourceResponse =
{std::make_shared<OCResourceResponse>()};
resourceResponse->setErrorCode(200);
resourceResponse->setResourceRepresentation(lightPtr->get(), DEFAULT_INTERFACE);
result = OCPlatform::notifyListOfObservers(  lightPtr->getHandle(),
lightPtr->m_interestedObservers,
resourceResponse);}
else{
result = OCPlatform::notifyAllObservers(lightPtr->getHandle());
}
if(OC_STACK_NO_OBSERVERS == result){
cout << "No More observers, stopping notifications" << endl;
gObservation = 0;}}}
return NULL;}
void * handleSlowResponse (void *param, std::shared_ptr<OCResourceRequest> pRequest){
// This function handles slow response case
LightResource* lightPtr = (LightResource*) param;
// Induce a case for slow response by using sleep
std::cout << "SLOW response" << std::endl;
sleep (10);
auto pResponse = std::make_shared<OC::OCResourceResponse>();
pResponse->setRequestHandle(pRequest->getRequestHandle());
pResponse->setResourceHandle(pRequest->getResourceHandle());
pResponse->setResourceRepresentation(lightPtr->get());
pResponse->setErrorCode(200);
pResponse->setResponseResult(OC_EH_OK);
// Set the slow response flag back to false
isSlowResponse = false;
OCPlatform::sendResponse(pResponse);
return NULL;}
```

```
void PrintUsage()
{std::cout << std::endl;
std::cout << "Usage : simpleserver <value>\n";
std::cout << "    Default - Non-secure resource and notify all observers\n";
std::cout << "    1 - Non-secure resource and notify list of observers\n\n";
std::cout << "    2 - Secure resource and notify all observers\n";
std::cout << "    3 - Secure resource and notify list of observers\n\n";
std::cout << "    4 - Non-secure resource, GET slow response, notify all observers\n";
}
static FILE* client_open(const char* /*path*/, const char *mode)
{return fopen("./oic_svr_db_server.json", mode);}
int main(int argc, char* argv[])
{PrintUsage();
OCPersistentStorage ps {client_open, fread, fwrite, fclose, unlink };
if (argc == 1)
{isListOfObservers = false;
isSecure = false;}
else if (argc == 2)
{int value = atoi(argv[1]);
switch (value){
case 1:
isListOfObservers = true;
isSecure = false;
break;
case 2:
isListOfObservers = false;
isSecure = true;
break;
case 3:
isListOfObservers = true;
isSecure = true;
break;
case 4:
isSlowResponse = true;
break;
default:
break;}}else{
return -1;}
// Create PlatformConfig object
PlatformConfig cfg {
OC::ServiceType::InProc,
OC::ModeType::Server,
"0.0.0.0", // By setting to "0.0.0.0", it binds to all available interfaces
0,         // Uses randomly available port
OC::QualityOfService::LowQos,
&ps};
OCPlatform::Configure(cfg);
try{
```

```
// Create the instance of the resource class
// (in this case instance of class 'LightResource').
LightResource myLight;
// Invoke createResource function of class light.
myLight.createResource();
std::cout << "Created resource." << std::endl;
myLight.addType(std::string("core.brightlight"));
myLight.addInterface(std::string(LINK_INTERFACE));
std::cout << "Added Interface and Type" << std::endl;
// A condition variable will free the mutex it is given, then do a non-
// intensive block until 'notify' is called on it.  In this case, since we
// don't ever call cv.notify, this should be a non-processor intensive version
// of while(true);
std::mutex blocker;
std::condition_variable cv;
std::unique_lock<std::mutex> lock(blocker);
std::cout <<"Waiting" << std::endl;
cv.wait(lock, []{return false;});
}catch(OCException &e){
std::cout << "OCException in main : " << e.what() << endl;
}// No explicit call to stop the platform.
// When OCPlatform::destructor is invoked, internally we do platform cleanup
return 0;}
```

I.1.2 Código Computador

```
#include <string>
#include <map>
#include <cstdlib>
#include <pthread.h>
#include <mutex>
#include <condition_variable>
#include "OCPlatform.h"
#include "OCApi.h"
using namespace OC;
typedef std::map<OCResourceIdentifier, std::shared_ptr<OCResource>> DiscoveredResourceMap;
DiscoveredResourceMap discoveredResources;
std::shared_ptr<OCResource> curResource;
static ObserveType OBSERVE_TYPE_TO_USE = ObserveType::Observe;
std::mutex curResourceLock;
class Light
{public:
bool m_state;
int m_power;
std::string m_name;
Light() : m_state(false), m_power(0), m_name("")
{}};
Light mylight;
int observe_count()
```

```
{static int oc = 0;
return ++oc;}
void onObserve(const HeaderOptions /*headerOptions*/, const OCRepresentation& rep,
const int& eCode, const int& sequenceNumber)
{try{
if(eCode == OC_STACK_OK && sequenceNumber != OC_OBSERVE_NO_OPTION)
{if(sequenceNumber == OC_OBSERVE_REGISTER)
{std::cout << "Observe registration action is successful" << std::endl;}
else if(sequenceNumber == OC_OBSERVE_DEREGISTER){
std::cout << "Observe De-registration action is successful" << std::endl;}
std::cout << "OBSERVE RESULT:"<<std::endl;
std::cout << "\tSequenceNumber: "<< sequenceNumber << std::endl;
rep.getValue("state", mylight.m_state);
rep.getValue("power", mylight.m_power);
rep.getValue("name", mylight.m_name);
std::cout << "\tstate: " << mylight.m_state << std::endl;
std::cout << "\tpower: " << mylight.m_power << std::endl;
std::cout << "\tname: " << mylight.m_name << std::endl;
if(observe_count() == 11)
{std::cout<<"Cancelling Observe..."<<std::endl;
OCStackResult result = curResource->cancelObserve();
std::cout << "Cancel result: "<< result <<std::endl;
sleep(10);
std::cout << "DONE"<<std::endl;
std::exit(0);}}
else{
if(sequenceNumber == OC_OBSERVE_NO_OPTION)
{std::cout << "Observe registration or de-registration action is failed" << std::endl;}
else{
std::cout << "onObserve Response error: " << eCode << std::endl;
std::exit(-1);}}}
catch(std::exception& e)
{std::cout << "Exception: " << e.what() << " in onObserve" << std::endl;}}
void onPost2(const HeaderOptions& /*headerOptions*/,
const OCRepresentation& rep, const int eCode)
{try{
if(eCode == OC_STACK_OK || eCode == OC_STACK_RESOURCE_CREATED)
{
std::cout << "POST request was successful" << std::endl;
if(rep.hasAttribute("createduri")){
std::cout << "\tUri of the created resource: "
<< rep.getValue<std::string>("createduri") << std::endl;
}else{
rep.getValue("state", mylight.m_state);
rep.getValue("power", mylight.m_power);
rep.getValue("name", mylight.m_name);
std::cout << "\tstate: " << mylight.m_state << std::endl;
std::cout << "\tpower: " << mylight.m_power << std::endl;
```

```
std::cout << "\tname: " << mylight.m_name << std::endl;
}
if (OBSERVE_TYPE_TO_USE == ObserveType::Observe)
std::cout << std::endl << "Observe is used." << std::endl << std::endl;
else if (OBSERVE_TYPE_TO_USE == ObserveType::ObserveAll)
std::cout << std::endl << "ObserveAll is used." << std::endl << std::endl;
curResource->observe(OBSERVE_TYPE_TO_USE, QueryParamsMap(), &onObserve);}
else{
std::cout << "onPost2 Response error: " << eCode << std::endl;
std::exit(-1);}}
catch(std::exception& e)
{
std::cout << "Exception: " << e.what() << " in onPost2" << std::endl;
}}
void onPost(const HeaderOptions& /*headerOptions*/,
const OCRepresentation& rep, const int eCode)
{try{
if(eCode == OC_STACK_OK || eCode == OC_STACK_RESOURCE_CREATED)
{std::cout << "POST request was successful" << std::endl;
if(rep.hasAttribute("createduri"))
{std::cout << "\tUri of the created resource: "
<< rep.getValue<std::string>("createduri") << std::endl;
}else{
rep.getValue("state", mylight.m_state);
rep.getValue("power", mylight.m_power);
rep.getValue("name", mylight.m_name);
std::cout << "\tstate: " << mylight.m_state << std::endl;
std::cout << "\tpower: " << mylight.m_power << std::endl;
std::cout << "\tname: " << mylight.m_name << std::endl;
}
OCRepresentation rep2;
std::cout << "Posting light representation..."<<std::endl;
mylight.m_state = true;
mylight.m_power = 55;
rep2.setValue("state", mylight.m_state);
rep2.setValue("power", mylight.m_power);
curResource->post(rep2, QueryParamsMap(), &onPost2);
}else{
std::cout << "onPost Response error: " << eCode << std::endl;
std::exit(-1);}}
catch(std::exception& e){
std::cout << "Exception: " << e.what() << " in onPost" << std::endl;
}}
// Local function to put a different state for this resource
void postLightRepresentation(std::shared_ptr<OCResource> resource)
{if(resource){
OCRepresentation rep;
std::cout << "Posting light representation..."<<std::endl;
```

```
mylight.m_state = false;
mylight.m_power = 105;
rep.setValue("state", mylight.m_state);
rep.setValue("power", mylight.m_power);
// Invoke resource's post API with rep, query map and the callback parameter
resource->post(rep, QueryParamsMap(), &onPost);}}
// callback handler on PUT request
void onPut(const HeaderOptions& /*headerOptions*/, const OCRepresentation& rep, const int eCode)
{try{
if(eCode == OC_STACK_OK)
{std::cout << "PUT request was successful" << std::endl;
rep.getValue("state", mylight.m_state);
rep.getValue("power", mylight.m_power);
rep.getValue("name", mylight.m_name);
std::cout << "\tstate: " << mylight.m_state << std::endl;
std::cout << "\tpower: " << mylight.m_power << std::endl;
std::cout << "\tname: " << mylight.m_name << std::endl;
postLightRepresentation(curResource);}
else{
std::cout << "onPut Response error: " << eCode << std::endl;
std::exit(-1);}}
catch(std::exception& e){
std::cout << "Exception: " << e.what() << " in onPut" << std::endl;
}}
// Local function to put a different state for this resource
void putLightRepresentation(std::shared_ptr<OCResource> resource)
{if(resource){
OCRepresentation rep;
std::cout << "Putting light representation..."<<std::endl;
mylight.m_state = true;
mylight.m_power = 15;
rep.setValue("state", mylight.m_state);
rep.setValue("power", mylight.m_power);
// Invoke resource's put API with rep, query map and the callback parameter
resource->put(rep, QueryParamsMap(), &onPut);}}
// Callback handler on GET request
void onGet(const HeaderOptions& /*headerOptions*/, const OCRepresentation& rep, const int eCode)
{try{
if(eCode == OC_STACK_OK)
{std::cout << "GET request was successful" << std::endl;
std::cout << "Resource URI: " << rep.getUri() << std::endl;
rep.getValue("state", mylight.m_state);
rep.getValue("power", mylight.m_power);
rep.getValue("name", mylight.m_name);
std::cout << "\tstate: " << mylight.m_state << std::endl;
std::cout << "\tpower: " << mylight.m_power << std::endl;
std::cout << "\tname: " << mylight.m_name << std::endl;
putLightRepresentation(curResource);
```

```
}else{
std::cout << "onGET Response error: " << eCode << std::endl;
std::exit(-1);}}
catch(std::exception& e){
std::cout << "Exception: " << e.what() << " in onGet" << std::endl;
}}
// Local function to get representation of light resource
void getLightRepresentation(std::shared_ptr<OCResource> resource)
{if(resource){
std::cout << "Getting Light Representation..."<<std::endl;
// Invoke resource's get API with the callback parameter
QueryParamsMap test;
resource->get(test, &onGet);}}
// Callback to found resources
void foundResource(std::shared_ptr<OCResource> resource)
{std::cout << "In foundResource\n";
std::string resourceURI;
std::string hostAddress;
try{{
std::lock_guard<std::mutex> lock(curResourceLock);
if(discoveredResources.find(resource->uniqueIdentifier()) == discoveredResources.end()){
std::cout << "Found resource " << resource->uniqueIdentifier() <<
" for the first time on server with ID: "<< resource->sid()<<std::endl;
discoveredResources[resource->uniqueIdentifier()] = resource;
}else{
std::cout<<"Found resource "<< resource->uniqueIdentifier() << " again!"<<std::endl;}
if(curResource){
std::cout << "Found another resource, ignoring"<<std::endl;
return;}}
// Do some operations with resource object.
if(resource){
std::cout<<"DISCOVERED Resource:"<<std::endl;
// Get the resource URI
resourceURI = resource->uri();
std::cout << "\tURI of the resource: " << resourceURI << std::endl;
// Get the resource host address
hostAddress = resource->host();
std::cout << "\tHost address of the resource: " << hostAddress << std::endl;
// Get the resource types
std::cout << "\tList of resource types: " << std::endl;
for(auto &resourceTypes : resource->getResourceTypes())
{std::cout << "\t\t" << resourceTypes << std::endl;}
// Get the resource interfaces
std::cout << "\tList of resource interfaces: " << std::endl;
for(auto &resourceInterfaces : resource->getResourceInterfaces())
{std::cout << "\t\t" << resourceInterfaces << std::endl;
}if(resourceURI == "/a/light"){
curResource = resource;
```

```
// Call a local function which will internally invoke get API on the resource pointer
getLightRepresentation(resource);}}
else{
// Resource is invalid
std::cout << "Resource is invalid" << std::endl;}
}catch(std::exception& e){
std::cerr << "Exception in foundResource: "<< e.what() << std::endl;
}}
void printUsage(){
std::cout << std::endl;
std::cout << "--------------------------------------------------------------------------------\n";
std::cout << "Usage : simpleclient <ObserveType>" << std::endl;
std::cout << "   ObserveType : 1 - Observe" << std::endl;
std::cout << "   ObserveType : 2 - ObserveAll" << std::endl;
std::cout << "--------------------------------------------------------------------------------\n\n";
}
void checkObserverValue(int value)
{if (value == 1){
OBSERVE_TYPE_TO_USE = ObserveType::Observe;
std::cout << "<===Setting ObserveType to Observe===>\n\n";
}else if (value == 2){
OBSERVE_TYPE_TO_USE = ObserveType::ObserveAll;
std::cout << "<===Setting ObserveType to ObserveAll===>\n\n";
}else{
std::cout << "<===Invalid ObserveType selected."
<<" Setting ObserveType to Observe===>\n\n";}}
static FILE* client_open(const char* /*path*/, const char *mode)
{return fopen("./oic_svr_db_client.json", mode);}
int main(int argc, char* argv[]) {
std::ostringstream requestURI;
OCPersistentStorage ps {client_open, fread, fwrite, fclose, unlink };
try{
printUsage();
if (argc == 1)
{std::cout << "<===Setting ObserveType to Observe and ConnectivityType to IP===>\n\n";
}else if (argc == 2){
checkObserverValue(std::stoi(argv[1]));}
else{
std::cout << "<===Invalid number of command line arguments===>\n\n";
return -1;}}
catch(std::exception& ){
std::cout << "<===Invalid input arguments===>\n\n";
return -1;}
// Create PlatformConfig object
PlatformConfig cfg {
OC::ServiceType::InProc,
OC::ModeType::Both,
"0.0.0.0",
```

```
0,
OC::QualityOfService::LowQos,
&ps
};
OCPlatform::Configure(cfg);
try{
// makes it so that all boolean values are printed as 'true/false' in this stream
std::cout.setf(std::ios::boolalpha);
// Find all resources
requestURI << OC_RSRVD_WELL_KNOWN_URI;// << "?rt=core.light";
OCPlatform::findResource("", requestURI.str(),
CT_DEFAULT, &foundResource);
std::cout<< "Finding Resource... " <<std::endl;
// Find resource is done twice so that we discover the original resources a second time.
// These resources will have the same uniqueidentifier (yet be different objects), so that
// we can verify/show the duplicate-checking code in foundResource(above);
OCPlatform::findResource("", requestURI.str(),
CT_DEFAULT, &foundResource);
std::cout<< "Finding Resource for second time..." << std::endl;
// A condition variable will free the mutex it is given, then do a non-
// intensive block until 'notify' is called on it.  In this case, since we
// don't ever call cv.notify, this should be a non-processor intensive version
// of while(true);
std::mutex blocker;
std::condition_variable cv;
std::unique_lock<std::mutex> lock(blocker);
cv.wait(lock);
}catch(OCException& e){
oclog() << "Exception in main: "<<e.what();}
return 0;}
```