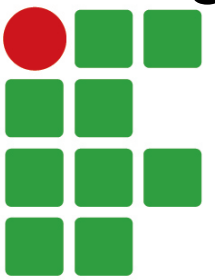
Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Câmpus Formosa
Tecnologia em Análise e Desenvolvimento de Sistemas

A Importância de Código Livre para Padronização da Comunicação entre Objetos Inteligentes no Contexto de Internet das Coisas: Um Estudo de Caso do Framework Iotivity



INSTITUTO FEDERAL
Goiás
Câmpus
Formosa

Autor: Jhuan Victor Spindola Pereira
Orientador: Me. Mario Teixeira Lemes

Formosa, GO
2018

Jhuan Victor Spindola Pereira

A Importância de Código Livre para Padronização da Comunicação entre Objetos Inteligentes no Contexto de Internet das Coisas: Um Estudo de Caso do Framework Iotivity

Monografia submetida ao curso de graduação em Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia de Goiás, câmpus Formosa, como requisito parcial para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Instituto Federal de Educação, Ciência e Tecnologia de Goiás

Câmpus Formosa

Orientador: Me. Mario Teixeira Lemes

Formosa, GO

2018

Jhuan Victor Spindola Pereira
A Importância de Código Livre para Padronização da Comunicação entre Objetos Inteligentes no Contexto de Internet das Coisas: Um Estudo de Caso do Framework Iotivity/ Jhuan Victor Spindola Pereira. – Formosa, GO, 2018-
95 p. : il. (algumas color.) ; 30 cm.

Orientador: Me. Mario Teixeira Lemes

Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Câmpus Formosa , 2018.

1. Internet das Coisas. 2. Padronização. I. Me. Mario Teixeira Lemes. II.Instituto Federal de Educação, Ciência e Tecnologia de Goiás. III. Câmpus Formosa. IV. A Importância de Código Livre para Padronização da Comunicação entre Objetos Inteligentes no Contexto de Internet das Coisas: Um Estudo de Caso do Framework Iotivity

CDU 02:141:005.6

Jhuan Victor Spindola Pereira

A Importância de Código Livre para Padronização da Comunicação entre Objetos Inteligentes no Contexto de Internet das Coisas: Um Estudo de Caso do Framework Iotivity

Monografia submetida ao curso de graduação em Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia de Goiás, câmpus Formosa, como requisito parcial para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Trabalho aprovado. Formosa, GO, 12 de dezembro de 2018:

Me. Mario Teixeira Lemes
Orientador

Esp. Murilo de Assis Silva
Convidado 1

Ma. Uyara Ferreira Silva
Convidado 2

Formosa, GO
2018

Resumo

A Internet das Coisas propõe que objetos do cotidiano se conectem à Internet e consigam comunicar-se entre si. Um dos desafios deste paradigma é o processo de comunicação, uma vez que deve ser possível o estabelecimento e a troca de informações entre dispositivos heterogêneos. O processo de padronização na comunicação, possível através da criação e uso de *middlewares* de código aberto, tem se mostrado um importante caminho para a consolidação e o efetivo uso de aplicações no contexto de Internet das Coisas. O *middleware Iotivity*, desenvolvido pela *AllJoin*, é um *framework* de código aberto que permite a conectividade entre diferentes dispositivos, o que torna possível a construção de aplicações e serviços destinados à Internet das Coisas. O objetivo deste Trabalho de Conclusão de Curso é realizar um estudo exploratório a cerca de Internet das Coisas e a padronização na construção de *middlewares* para comunicação de objetos inteligentes. O *Framework Iotivity*, da *AllJoin*, será o objeto de estudo para confecção de um material educacional gratuito, o que permitirá maior adesão da comunidade científica, bem como norteará novos pesquisados interessados no uso e construção de aplicações inerentes ao contexto de Internet das Coisas. Um estudo de caso relacionado a construção de um sistema inteligente de detecção e divulgação de erupção vulcânica é apresentado no contexto de comunicação entre objetos inteligentes heterogêneos. A padronização da comunicação em Internet das Coisas e o desenvolvimento de software livre colaborativo na construcão de *middlewares* é um importante passo para a consolidação do paradigma tal como é concebido.

Palavras-chaves: Internet das Coisas. Padronização. Comunicação.

Abstract

Internet of Things proposes everyday objects to connect to the Internet and communicate with each other. One of the challenges of this paradigm is the communication process, since it should be possible to establish and exchange information between different devices. The process of standardization in communication, which is possible through creation and use of open source middleware, a result has an important path for the consolidation and effective use of applications in the context of the Internet of Things. O middleware Iotivity, developed by AllJoin, is an open source framework that connectivity between different devices, which makes it possible to construct of applications and services treated to the Internet of Things. The objective of this work is to conduct an exploratory study on the Internet of Things and a standardization in the construction of middleware for intelligent object communication. O AllJoin's framework activity will be the object of study for making a material free education, which is greater adherence of the scientific community, as well as will guide new researches interested in the use and construction of applications Internet context of Things. A case study related to the construction of a detection and dissemination of volcanic eruptions is presented in the context of between heterogeneous intelligent objects. A standardization of communication on the Internet and the development of free collaborative software in construction middleware is an important step towards consolidating the paradigm as it is designed.

Key-words: Internet of Things. Standardization. Communication.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IoT	Internet of Things
RFID	Radio-Frequency Identification
NFC	Near Field Communication
IP	Internet Protocol
Wi-Fi	Wireless Fidelity
FPGA	Field Programmable Gate Array
SBC	Single Board Computer
SDK	Software Development Kit
OSI	Open Source Initiative
PaaS	Platform as a Service
M2M	Machine to Machine
API	Application Programming Interface
SRM	Security Resource Manager
SVR	Secure Virtual Resource
ACL	Access Control List
OIC	Open Interconnect Consortium
BNDES	Banco Nacional de Desenvolvimento Econômico e Social

Sumário

1 Introdução

A Internet das Coisas (*Internet of Things* - IoT) é uma ampliação da Internet atual, que abre oportunidade para que objetos do cotidiano consigam se conectar a Internet e se comunicarem uns com os outros, abrindo um vasto leque para novas aplicações (SANTOS et al., 2016).

O conceito tradicional de redes de computadores não é aplicável em IoT devido a grande quantidade de equipamentos e tecnologias de comunicação conectados à Internet. Objetos inteligentes contribuem para a confirmação desse conceito. Segundo (TANENBAUM, 2002), rede de computadores é um conjunto de computadores autônomos interconectados por uma única tecnologia. Nota-se que a definição realizada por Tanenbaum não pode ser aplicada a IoT devido a grande heterogeneidade apresentada neste paradigma.

IoT se desenvolve a partir dos avanços de outras áreas, tais como sistemas embarcados, microeletrônica, comunicação e sensoriamento, e por isso pode ser considerada como uma combinação de diversas tecnologias que se complementam para viabilizar a integração dos objetos ao mundo virtual.

O desenvolvimento da IoT abre um leque para a criação de novas aplicações, a citar: cidades inteligentes, automação de ambientes e sistemas de coleta e organizacão de dados para diagnósticos médicos. Por este motivo, IoT é um nicho de pesquisa, com diversos desafios de pesquisa em aberto, para profissionais da área acadêmica e industrial, além de ter sido reconhecida pela Gartner uma tecnologia emergente (GARTNERGROUP, 2018).

IoT possui diversos desafios pesquisa e que possuem um papel importante na sua evolução. Dentre esses desafios, destacam-se: de que forma os dispositivos inteligentes são identificados na rede, o desempenho e eficiência dos objetos inteligentes, segurança, interoperabilidade, escalabilidade e a criação e implementação de padrões os processo de comunicação.

A padronização de protocolos para a comunicação entre objetos inteligentes é um fator essencial para sua consolidação: dispositivos, com capacidades computacionais e sistemas operacionais heterogêneos, devem se conectar entre si e trocar informações ininterruptamente. Protocolos e *middlewares* padronizados e adaptáveis possibilitam a integração e a consolidação dos objetos inteligentes, permitindo a concepção e implementação de IoT como a mesma é concebida.

De acordo com (NGUYEN MARYLINE LAURENT B, 2015), o desafio da comunidade científica se resume em como realizar adaptações nos protocolos convencionais utilizados na Internet para o projeto, *design* e o desenvolvimento de aplicações aplicáveis

ao cenário de IoT, visto que a heterogeneidade dos objetos deve ser considerada para que seja possível dispositivos computacionais com capacidades computacionais limitados se comunicarem àqueles mais robustos em termos computacionais.

Soluções livres e de código aberto para os problemas de comunicação de objetos inteligentes no contexto de IoT e o desenvolvimento de *frameworks* são essenciais para o desenvolvimento e a consolidação deste paradigma. Código aberto, ou *open source*, é um modelo de desenvolvimento que permite qualquer indivíduo e/ou empresa consultar, examinar ou modificar o produto, com licenciamento livre no *design*, esquema e distribuição.

Frameworks de código aberto colaboram com o desenvolvimento de IoT, expande e aprimora suas funcionalidades, aumentando sua flexibilidade e compatibilidade com outros dispositivos, algo fundamental em um paradigma que propõe a possibilidade de objetos inteligentes distintos se comunicarem ininterruptivelmente entre si.

Iotivity é um *framework* de código aberto, patrocinado pela *Open Connectivity Foundation* (OCF), criado para reunir a comunidade de desenvolvedores com o objetivo de acelerar o desenvolvimento da estrutura e os serviços necessários para solucionar os desafios de pesquisa em IoT, especialmente o que se refere a padronização da comunicação entre objetos inteligentes. Este projeto tem como objetivo garantir que objetos inteligentes possam se conectar, com segurança e confiabilidade à Internet e uns aos outros, e trocar informações, endereçando o desafio de comunicação entre objetos inteligentes heterogêneos.

Além dos projetos de código aberto que colaboram com a expansão, evolução e consolidação de IoT, existe outro fator importante para que essa expansão aconteça: a confecção e divulgação de materiais educacionais livres. Esses materiais são essenciais para ajudar no aprendizado de pessoas que estão interessadas em adquirir e aprofundar conhecimentos em uma determinada área. Iotivity, por se tratar de um projeto colaborativo, é marcada pela limitação da quantidade de materiais educacionais livres, o que pode ser um fator impeditivo no aprofundamento e no uso deste *middleware* para o desenvolvimento de aplicações de IoT.

A metodologia de pesquisa que será adotada neste trabalho é a pesquisa aplicada exploratória e a realização de um estudo de caso. O principal objetivo com a pesquisa exploratória é "desencadear um processo de identificação que permita a identificação da natureza do fenômeno e aponte as características essenciais das variáveis que se quer estudar (KOCHE, 2011).

O planejamento da pesquisa exploratória baseará, em um primeiro momento, na realização da pesquisa bibliográfica acerca de livros, artigos científicos, trabalhos de conclusão de curso (TCC), dissertações de mestrado, teses de doutorado, anais e meios eletrônicos sobre conceitos relacionados ao paradigma de IoT, plataformas de dispositivos

embarcados, tecnologias de comunicação e de aplicações possíveis de serem construídas considerando limitações de processamento, memória e energia típicas dos dispositivos IoT bem como *middlewares* e plataformas aplicacáveis a IoT, especialmente a iniciativa de código aberto Iotivity.

Bases eletrônicas, como "SCOPUS", "IEEE Xplore Digital Libray", "ACM Digital Library", sites de busca e repositórios de Universidades serão utilizadas para construção do acervo bibliográfico necessário para a realização da pesquisa. Em um segundo momento será realizado um estudo de caso exploratório-descritivo acerca do *framework* Iotivity, com o intuito de realizar um estudo aprofundado para seu amplo e detalhado conhecimento.

Portanto, o objetivo deste Trabalho de Conclusão de Curso (TCC) é realizar um estudo exploratório sobre IoT, especialmente sobre a iniciativa da Iotivity e confeccionar um material educacional livre a respeito deste *framework* para divulgação a toda comunidade científica. Para atingir este objetivo será realizado um estudo de caso de comunicação entre dispositivos inteligentes, a partir do uso do *framework* de código aberto, mostrando como que diferentes objetos inteligentes podem se beneficiar, através do uso do *framework*, para estabelecimento da conexão e dos processos de comunicação. O desenvolvimento deste trabalho ajudará a sanar a escassez de materiais didáticos e colaborará com a expansão e consolidação de IoT, especialmente no tocante a materiais livres e de código-aberto.

Este TCC está dividido em 4 (quatro) capítulos. Neste capítulo foi apresentada uma introdução sobre IoT, os principais desafios de pesquisa, a filosofia de código aberto e sua importância no processo de padronização da comunicação entre os objetos inteligentes no contexto de IoT. O Capítulo 2 abordará um referencial teórico sobre IoT, blocos básicos de construção, tipos de arquitetura e os principais *middlewares* aplicáveis a este cenário existentes na literatura. O Capítulo 3 trará tutoriais de como habilitar/construir o *framework* da Iotivity em diferentes dispositivos inteligentes juntamente com explicações de recursos obtidos ao se utilizar o *framework da Iotivity*. O Capítulo 4 é responsável por trazer o estudo de caso do *framework Iotitvity*, através do desenvolvimento de um sistema inteligente de detecção e divulgação de erupção vulcânica. Por fim no Capítulo 5 tem-se as considerações finais e a direção de possíveis trabalhos futuros.

2 Referencial Teórico

2.1 Introdução

Neste capítulo são apresentadas conceitos relacionados a IoT, tais como a origem do termo e exemplos de aplicações, além de plataformas computacionais comuns neste contexto. São abordadas ainda a arquitetura básica necessária para construção e o desenvolvimento de aplicações, bem como os principais *middlewares* que colaboram com o processo de desenvolvimento e o fornecimento de serviços úteis aos usuários finais.

2.2 História de Criação do Termo IoT

O termo IoT foi criado em setembro de 1999 por Kevin Ashton, um tecnológico britânico que criou um sistema de sensores omnipresentes que conectava o mundo físico à Internet, enquanto trabalhava com Identificação por Rádio Frequência (RFID) (FUTUREWEI, 2018).

Ashton explicou o potencial de IoT em uma entrevista da seguinte forma:

> Os computadores de hoje – e, portanto, a Internet – são quase totalmente dependentes de seres humanos para informação. Quase todos os cerca de 50 *petabytes* (um *petabyte* é 1.024 *terabytes*) de dados disponíveis na Internet foram capturados e criados por seres humanos, digitando, pressionando um botão de gravação, tirando uma foto digital ou digitalizando um código de barras. O problema é, as pessoas têm tempo, atenção e precisão limitados – o que significa que eles não são muito bons em captar dados sobre as coisas no mundo real. Se tivéssemos computadores que soubessem de tudo que se pode saber sobre as coisas, os dados poderiam ser reunidos sem qualquer ajuda de nós – isso seria capaz de monitorar e contar tudo e reduzir significativamente o desperdício, perda e custo. Gostaríamos de saber quando as coisas precisarem de substituição ou reparação ou registrar se algo está fresco ou pronto para consumo (FINEP, 2015).

Já segundo Mark Weiser, "As tecnologias mais importantes são aquelas que desaparecem. Elas se integram à vida do dia a dia, ao nosso cotidiano e tornam-se indistinguíveis" (WEISER, 1999). Essa idéia simplifica o motivo que IoT é considerada uma tecnologia emergente. Note na Figura 1 a quantidade de objetos inteligentes que podem ser agregadas ao paradigma de IoT. Qualquer objeto pode tornar-se inteligente e oferecer serviços úteis aos usuários.

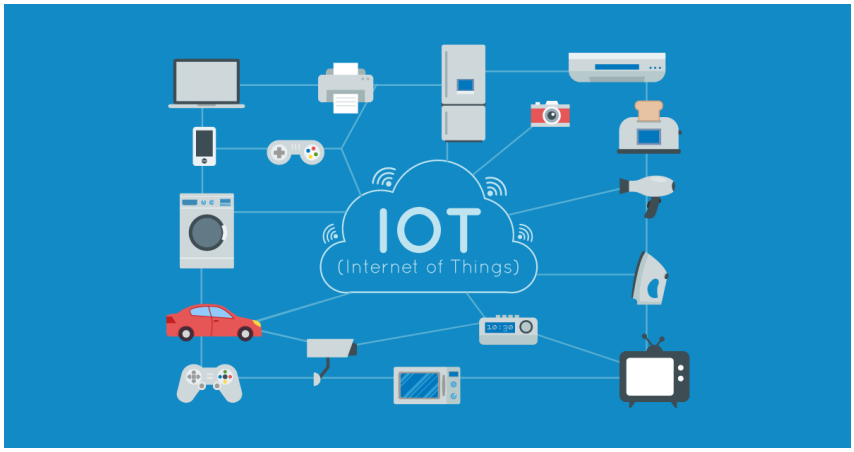


Figura 1 – Ilustração do paradigma de IoT

2.2.1 Exemplos de Aplicações

IoT pode ser observada em diferentes áreas de aplicação. Na agricultura é possível realizar o monitoramento da irrigação nas plantações do campo e com dados precisos da quantidade de água gasta, umidade da terra, umidade do ar e temperatura. Em parceria com a Usina São Martinho, de Pradópolis (SP), e com apoio do BNDES (Banco Nacional de Desenvolvimento Econômico e Social - BNDES), o CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações - CPqD) instalou sensores em máquinas agrícolas, que enviam dados sobre posição e desempenho dos equipamentos, via rádio em frequência de 250 MHZ para as estações instaladas nas torres da fábrica. Os dados sobre equipamentos como tratores e caminhões são enviados para uma plataforma que pode ser acessada por meio de dispositivos móveis. Na Figura 2 é possível observar como é a estrutura do monitoramento de um dos pivôs de irrigação.(NEGOCIOS/GLOBO, 2017)

Figura 2 – Estrutura do Monitor de Irrigação

A CERTI (Fundação Centros de Referência de Tecnologias Inovadoras - CERTI),

de Santa Catarina, vem trabalhando em uma solução para fabricação de itens personalizados de maneira eficiente e com baixo custo. O sistema possibilita enviar remotamente, as configurações necessárias para que a máquina personalize um determinado item.(NEGOCIOS/GLOBO, 2017)

IoT também pode ser observada na área da saúde. O sistema, *Vaccine Smart Fridge* (ICLINIC, 2018), possibilita o transporte de vacinas que necessitam de sistema de refrigeração. Neste caso são utilizados sensores, localizados na geladeira portátil e conectados em uma plataforma IoT, tornando possível o monitoramento do estado, em tempo real, das vacinas. Note na Figura 3 a estrutura do sistema *Vaccine Smart Fridge*.

Figura 3 – Vaccine Smart Fridge

2.3 Blocos Básicos de IoT

IoT pode ser definida como uma combinação de várias tecnologias que possibilitam a integração dos objetos do dia-a-dia ao mundo virtual. Para o endereçamento de IoT é necessário endereçar os 6 (seis) blocos básicos de construção, a citar: Identificação, Sensores/Atuadores, Comunicação, Computação, Serviços e Semântica (SANTOS et al., 2016). Note na Figura 4 a arquitetura básica presente no paradigma de IoT.

2.3.1 Identificação

Protocolos de Identificação são responsáveis por trazer soluções de identificação única aos objetos inteligentes. Alguns exemplos de tecnologias que possibilitam os processos de identificação são: RFID (FUTUREWEI, 2018) e endereçamento IP (*Internet Protocol* - IP) (IETF, 1981).

A tecnologia de RFID é um método de identificação automática que utiliza sinais de radio, recuperando e armazenando dados remotamente através de etiquetas RFID.

Uma etiqueta RFID é um *transponder* capaz de responder a sinais de rádio através de antenas e *chips* de silício.

Já o endereçamento IP é um rótulo atribuído a todos os dispositivos que se conectam a uma rede de computadores e que utilizam o protocolo de Internet para comunicação. O IP pode ser usado para identificação de interface de hospedeiro ou para rede e endereçamento de localização.

2.3.2 Sensores/Atuadores

Sensores, no contexto de IoT, são dispositivos computacionais capazes de coletar informações sobre o contexto onde os objetos estão localizados, para em seguida enviar esses dados para serem armazenados em nuvem, data *warehouse* ou centros de armazenamento. Já os atuadores manipulam o ambiente e tomam decisões, a depender da aplicação, de acordo com os dados lidos pelos sensores.

2.3.3 Comunicação

Os processos de comunicação são relacionados com as tecnologias de comunicação utilizadas para conectar os objetos inteligentes e são responsáveis por grande parte do consumo de energia dos dispositivos computacionais, o que tornam-o um fator crítico. As tecnologias comumente usadas para comunicação entre objetos inteligentes são o Wi-Fi (IETF, 2009), Bluetooth (IEEE, 2018), RFID (FUTUREWEI, 2018), Comunicação por Campo de Proximidade (*Near Field Communication* - NFC) (ISO, 2013), Ethernet (PLUMMER, 1982), ZigBee (ALLIANCE, 2002), 3G/4G (KOODLI, 2011) e LoraWan (FARRELL, 2018).

Wi-Fi é uma rede sem fio (*wireless*), que torna possível o acesso à internet apenas por sinal de ondas de rádio, sem a necessidade de usar fios conectores. Já o *bluetooth* é uma especificação de rede sem fio, que provê uma maneira de trocar informações entre dispositivos, tais como celulares, *notebooks*, computadores, impressoras, câmeras digitais e consoles de videogames.

O RFID permite a comunicação entre objetos inteligentes através de etiquetas que transmitem a informação a partir da passagem por um campo de indução. A *Ethernet* é uma arquitetura de interconexão aplicáveis a redes de áreas locais (*Local Area Network* - LANs).

NFC é uma tecnologia que permite a troca de informações sem fio, entre dispositivos compatíveis, que estejam próximos uns dos outros. A conexão é estabelecida automaticamente assim que os dispositivos estejam próximos. Qualquer dispositivo que tenha um *chip* NFC pode utilizar essa tecnologia, como por exemplo, telefones celulares, *tablets*, crachás, pulseiras ou cartões de bilhetes eletrônicos.

A rede conhecida como ZigBee, criada pelo IEEE em conjunto com a ZigBee Alliance, foi criada com o intuito de disponibilizar uma rede com extrema baixa potência de operação, ocasionando um baixo consumo de energia nos dispositivos, estendendo a vida útil de suas baterias. Já a tecnologia 3G/4G utiliza-se da frequência das telefonias móveis para fornecer conectividade de Internet sem fio.

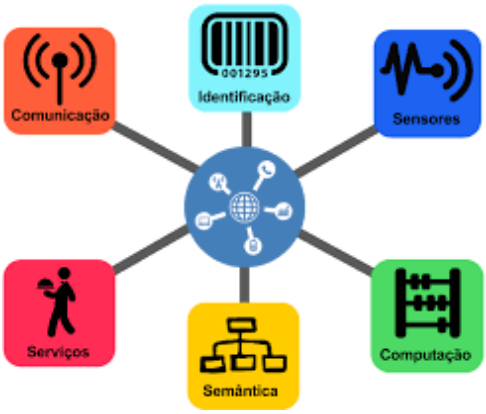


Figura 4 – Blocos básicos de construção da IoT

2.3.4 Computação

Bloco no qual estão incluídas as unidades de processamento, responsáveis por executar instruções (algoritmos) nos objetos inteligentes. Os microcontroladores, processadores e o Arranjo de Portas Programável em Campo (*Field Programmable Gate Array* - FPGA) (ISO, 2013) são exemplos de dispositivos utilizados nos processos computacionais em IoT.

Processadores e microcontroladores são formados por circuito integrados que possuem um núcleo de processamento e que podem ser programados para realizarem tarefas especificas, comumente utilizados em sistemas embarcados. Veja na Figura 5 uma representacão dos microcontroladores.

FGPA é um circuito integrado reprogramável, comumente utilizado para processamento de informações digitais. Consiste de um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado. Cada célula contém capacidade computacional para implementar funções lógicas e realizar roteamento para comunicação entre elas.

2.3.5 Serviços

São os serviços que podem ser providos pelas aplicações. Existem diversos tipos de serviços, tais como serviços de identificação que mapeiam informações coletadas em dados para o usuário, como por exemplo a umidade contida na terra de uma estufa.

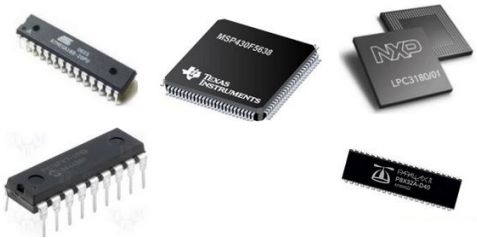


Figura 5 – Microcontroladores

Já os serviços de agregação de dados são aqueles responsáveis pela coleta dos dados homogêneos/heterogêneos, obtidos de diferentes objetos inteligentes. Serviços de Colaboração e Inteligência são utilizados como suporte para a tomada de decisões.

2.3.6 Semântica

Relaciona-se com a capacidade de extração de conhecimento dos objetos na IoT e o uso eficiente dos recursos existentes sobre os dados coletados para provimento de um determinado serviço. As técnicas que normalmente são usadas para identificação da semântica são o *Resource Description Framework* (SWARTZ, 2004), a *Web Ontology Language* (W3C, 2012) e a *Efficient XML Interchange* (W3C, 2016).

2.4 Arquitetura Básica dos Dispositivos em IoT

Os objetos inteligentes seguem uma arquitetura básica, composta por uma unidade de sensor/atuador, unidade de processamento/memória, unidade de comunicação e unidade de alimentação/energia (SANTOS et al., 2016). Note na Figura 6 a arquitetura básica dos dispositivos inteligentes presentes em IoT.

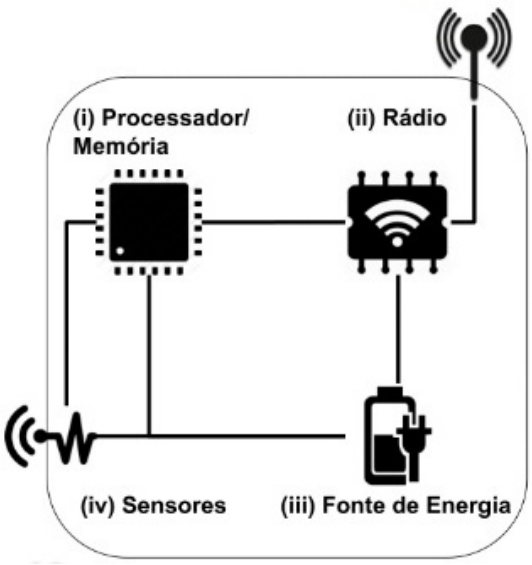


Figura 6 – Arquitetura Básica dos Dispositivos IoT

2.4.1 Unidade de Processamento/Memória

Possui um microcontrolador, responsável pelo processamento da informação, e uma memória para armazenamento de dados, bem como um conversor digital para conversão dos dados recebidos pelos sensores As CPUs usadas nessa unidade, são geralmente os mesmos utilizados em sistemas embarcados, que por sua vez não apresentam alta capacidade computacional.

2.4.2 Unidade de Sensor/Atuador

Essa unidade realiza o monitoramento de alguma variável, no caso de um sensor, ou uma perfaz uma determinada ação, se for um atuador. Os sensores coletam valores de grandezas físicas, tais como temperatura, umidade, movimento, dentre outras. Atuadores modificam o ambiente onde estão localizados, atendendo à comandos que podem ser manuais, elétricos e/ou mecânicos. A Figura 7 ilustra alguns sensores de determinadas grandezas físicas.

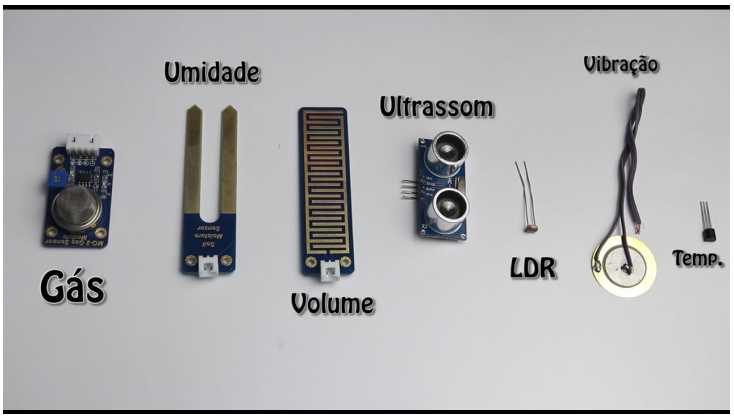


Figura 7 – Sensores

2.4.3 Unidade de Comunicação

A unidade de comunicação dos dispositivos inteligentes é composta por um ou mais canais de comunicação com ou sem fio. As tecnologias mais usadas para realizar a comunicação de objetos na IoT podem ser encontradas na Seção 2.3.3.

2.4.4 Fonte de Energia

Encarregada de alimentar todos os componentes do objeto inteligente, normalmente essa unidade é composta por uma bateria e um conversor ac-dc, capaz de transformar energia DC de baterias e pilhas em energia elétrica alternada, geralmente na tensão de 127 V ou 220 V.

2.5 Plataformas computacionais utilizadas em IoT

2.5.1 Raspberry Pi

Raspberry Pi, desenvolvido pel Fundação Raspberry Pi, é um minicomputador (*Single-Board Computer* - SBC), no qual todo o *hardware* é integrado em uma única placa. O Raspberry Pi possui o tamanho de um cartão de crédito e pode se conectar a monitores, teclados e *mouses*. É compatível com sistemas operacionais baseados em GNU/Linux e Windows 10 IoT. Na figura 8 é apresentado o Raspberry Pi 3 Modelo B+.

Figura 8 – Raspberry Pi 3 Modelo B+

2.5.2 Arduino

O arduino foi criado em 2005 por um grupo de 5 (cinco) pesquisadores, a citar: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis. O objetivo desses pesquisadores era desenvolver um dispositivo que fosse barato, funcional e fácil de se programar, o que o tornaria acessível para principiantes e profissionais. O Arduíno é composto por uma placa que possui um microcontrolador Atmel, circuitos de entrada/saída e que pode ser facilmente conectada à um computador. Possui uma linguagem padrão que é essencialmente C/C++ (THOMSEN, 2014). Note na Figura 9 o Arduíno, em sua versão UNO.

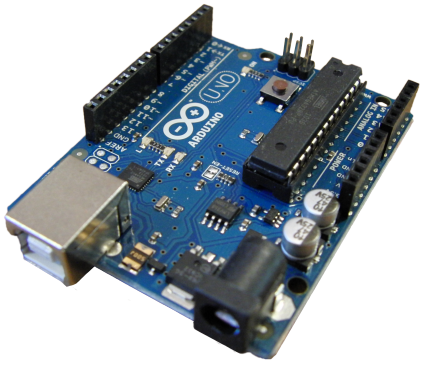


Figura 9 – Arduino Uno

2.6 *Middlewares* e Plataformas para IoT

2.6.1 EcoDIF (PIRES, 2012)

A EcoDIF foi proposta como grupo de trabalho na RNP (Rede Nacional de Ensino e Pesquisa - RNP) em 2012. Este projeto tem como objetivo desenvolver uma plataforma *Web* para conectar dispositivos e produtos com aplicações e/ou usuários finais, a fim de fornecer funcionalidades de controle, visualização, processamento e armazenamento de dados.

O intuito da EcoDIF é atuar como núcleo de um ecossistema IoT, oferecendo serviços (de *software*) focados: (i) na conectividade entre dispositivos e a Internet; (ii) em serviços de aplicação e (iii) em serviços de apoio. A EcoDIF pode atuar em aplicações de monitoramento ambiental, monitoramento de infraestrutura pública (transito, condições da estrada) e para compartilhamento de sensoriamento. Note na Figura 10, a visão geral do funcionamento plataforma EcoDIF.

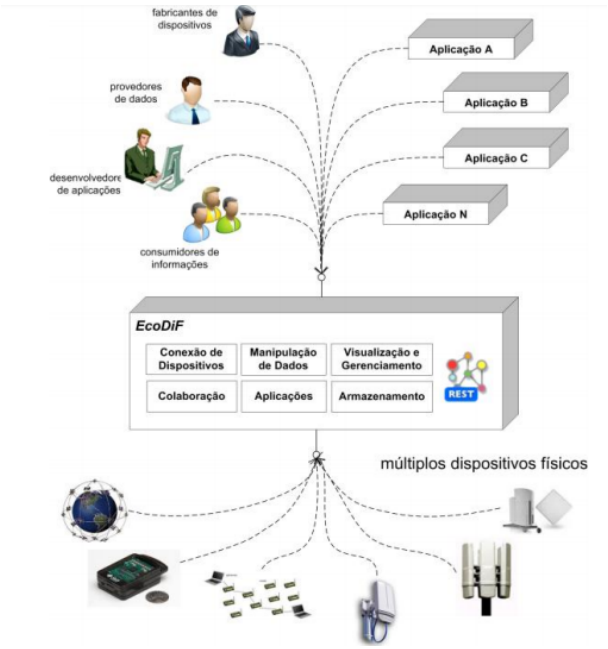


Figura 10 – Visão Geral da EcoDIF

2.6.2 Google Cloud IoT (CLOUD, 2018)

Google Cloud IoT é uma plataforma inteligente de IoT que desvenda os *insights* de negócios oferecidos pela sua rede global de dispositivos. Essa plataforma é um conjunto de serviços totalmente gerenciados e integrados para que seja possível, gerenciar e processar dados de dispositivos IoT coletados em larga escala, podendo também analisar esses dados em tempo real, implementar mudanças operacionais e realizar ações necessárias.

Google Cloug IoT não necessita de uma infraestrutura robusta para análise e armazenamento dos dados, uma vez que todos seus dispositivos são acomodados na nuvem, através dos protocolos do *Cloud IoT Core*. Os recursos ofertados pela Google são segurança de ponta-a-ponta, sistema global único, serviços integrados, análises de dados avançadas, infraestrutura gerenciável e otimização dos processos de negócio. A Figura 11 ilustra o fluxo de gerenciamento de dados no *Google Cloud IoT*.

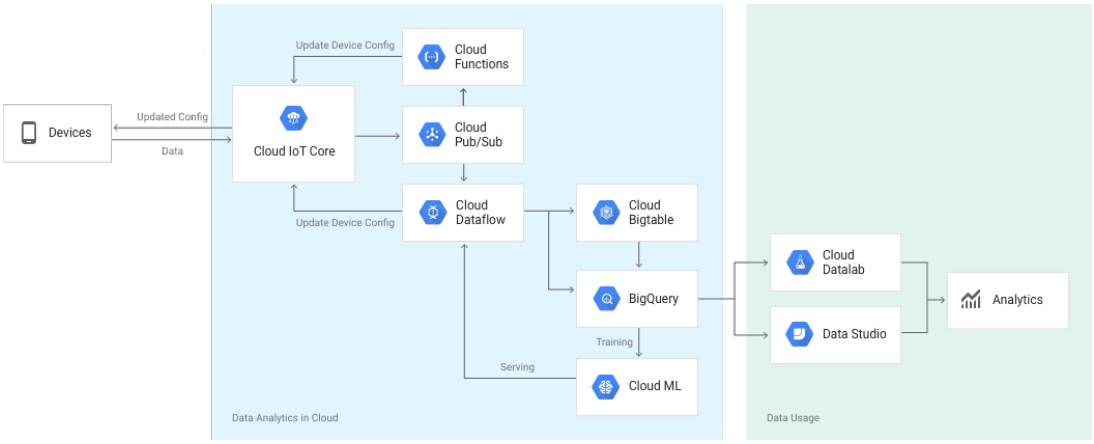


Figura 11 – Visão geral do gerenciamento de dados no Google Cloud IoT

2.6.3 Carriots (ALTAIR, 2017)

A Carriots, da *Altair Engineering*, é uma plataforma IoT inteligente como serviço (*Platform as a Service* - PaaS), feita para projetos de máquina a máquina (*Machine to Machine* - M2M). A plataforma possui recursos para acelerar o desenvolvimento de aplicativos IoT, além de oferecer escalabilidade simples à medida do crescimento dos projetos e dispositivos. A Carriots possibilita que os usuários armazenem dados de dispositivos conectados e implantem protótipos a milhares de dispositivos.

Os recursos oferecidos pela plataforma são gerenciamento de dispositivo, ouvintes, regras, acionadores, *engine* de aplicativo SDK, exportação de dados, alarmes personalizados, *debug and logs*, níveis de hierarquia customizáveis, gerenciamento de chaves de API, gerenciamento de usuários e painel de controle personalizado. Note na Figura 12 o funcionamento da plataforma Carriots.

2.6.4 LinkSmart (LINKSMART, 2018)

LinkSmart é um *framework* e uma infraestrutura que disponibiliza serviços para o desenvolvimento de aplicativos distribuídos na IoT. A plataforma é composta por vários componentes organizados em 3 (três) subprojetos:

1. ***LocalConnect:*** Fornece componentes para a criação de ambientes inteligentes locais que consistem em vários dispositivos, aplicativos e serviços, que podem ser

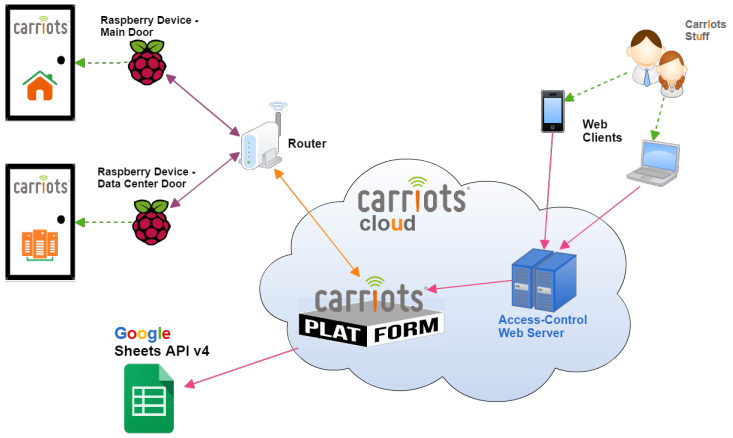


Figura 12 – Visão geral da plataforma Carriots

descobertos e comunicados com o uso da mensagem de publicação/assinatura ou solicitação/resposta.

2. ***GlobalConnect:*** Responsável pela conexão remota de ambientes *LinkSmart* pela Internet, permitindo a comunicação transparente de aplicativos e serviços, por meio de uma rede privada.

3. ***Services:*** Projeto para serviços de *middleware* que oferece funcionalidade estendida para implementação de aplicativos e plataformas de IoT.

2.6.5 OpenIoT (OPENIOT, 2018)

A OpenIoT possui um *middleware* de código aberto que obtêm informações de nuvens de sensores, sem se preocupar com a quantidade exata de sensores que são usados, explorando maneiras eficientes de se usar e gerenciar ambientes de nuvem para recursos IoT, como sensores e atuadores.

O *middleware* da OpenIoT é útil para diversas áreas cientificas e tecnológicas inter-relacionadas, abrangendo: *middleware* para sensores e redes de sensores, **ontologias** que são modelos semânticos e anotações para representar objetos conectados à Internet, juntamente com técnicas semânticas de dados abertos, incluindo esquemas de segurança e privacidade baseados em utilidade (*Cloud/Utility computing*).

2.6.6 IoTivity (JOIN, 2018)

A Iotivity é um projeto de código aberto, patrocinado pela OCF (*Open Connectivity Foundation* - OCF), criado para reunir a comunidade de código aberto para acelerar o desenvolvimento da estrutura e os serviços necessários para enfrentar os desafios na IoT.

Este projeto tem como objetivo de garantir que os objetos inteligentes possam se conectar com segurança e confiabilidade à internet e uns aos outros, na tentativa de criar

padrões para a comunicação desses objetos para que a IoT se concretize. O *middleware* Iotivity é foco de estudo desde TCC e é explorado no Capítulo 3.

2.7 Considerações Finais

Dispositivos inteligentes são aqueles que possuem recursos computacionais e podem se conectar à Internet para troca de informações. Os recursos computacionais dos dispositivos inteligentes, tais como raspberry pi e arduino, são limitados. *Middlewares* são soluções de *software* desenvolvidas para a padronização na comunicação, o que possibilita que diferentes dispositivos se comuniquem. O uso de *middlewares* de código-aberto, tais como o Iotivity, representa uma possibilidade de padronização nos processos de comunicação entre dispositivos inteligentes heterogênos.

3 O Framework Iotivity

3.1 Introdução

A Iotivity disponibiliza em seu site[1] uma *wiki* que faz parte da documentação do *framework*. Nesta *wiki* estão listadas grande parte das funcionalidades e capacidades do *middleware*, além de guias que orientam como realizar alterações e propor mudanças para serem incorporados ao *framework*.

Processos de documentação de *software* fornecem todas as informações importantes para o uso de um determinado *software*, podendo ser utilizada por usuários técnicos e/ou usuários finais. Uma boa documentação deve ser concisa, clara e objetiva. No caso da Iotivity, a documentação de *software* se torna ainda mais importante por ser uma plataforma *open source*, desenvolvida de forma colaborativa, no qual novas funcionalidades, correções e melhorias são implementadas constantemente, tornando necessário uma documentação robusta e atual.

O conteúdo da *wiki* é disponibilizado[2] em 3 (três) seções principais, a citar: preparando-se para desenvolver, guia de programação e notas técnicas. Dentro dessas seções existem tópicos com informações relevantes sobre o *framework*. Nas subseções a seguir serão listadas e descritas os tópicos da *wiki*, contemplando tutoriais de consrução em diferentes sistemas operacionais e explicação de suas principais técnicas/funcionalidades.

3.2 Como habilitar/construir o *Framework* Iotivity

3.2.1 GNU/Linux Ubuntu

Este tutorial realiza um passo-a-passo para construção do projeto Iotivity no Sistema Operacional GNU/Linux Ubuntu LTS 14.04. Note que alguns dos procedimentos descritos podem se alterar em outras versões do referido sistema operacional.

1º Passo - Instalação do git: Primeiramente, deve-se instalar o *git*, um *software* de gerenciamento de código-fonte, necessário para obter acesso ao código fonte da Iotivity. Para instalá-lo, use o seguinte comando no terminal, conforme demonstrado na Figura 13.

$ sudo apt-get install git-core

2º Passo - Instalação do scons e g++: Em seguida instale o scons, uma

[1] https://wiki.iotivity.org
[2] Atualização mais recente em 20/12/2017

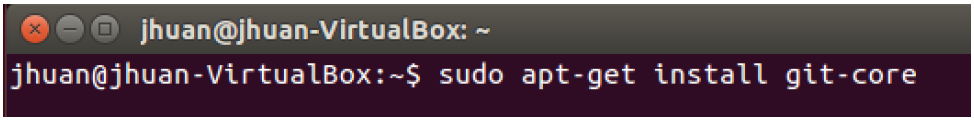


Figura 13 – Instalação do *software* git

ferramenta de compilação multiplataforma, que auxiliará no processo de compilação do *framework*.

$ sudo apt-get install scons

A biblioteca g++ é necessária para criação da pilha do *framework* Iotivity. Realize a instalação dos pacotes build-essential e g++ conforme Figura 14.

$ sudo apt-get install build-essential g++

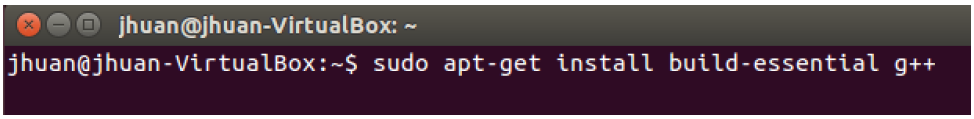


Figura 14 – Instalação do g++

3º Passo - Instalação do SSH: Para se conectar ao repositório da Iotivity é utilizado o protocolo *Secure Shell* (SSH). Para instalar o SSH, utilize o comando no terminal de acordo com a Figura 15.

$ sudo apt-get install ssh

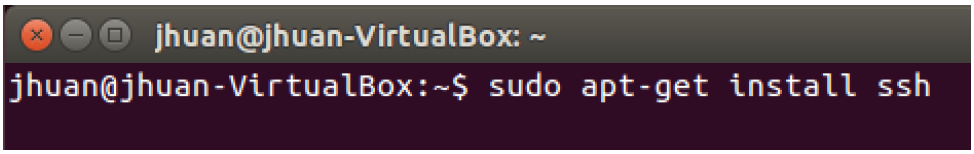


Figura 15 – Instalação do SSH

4º Passo - Instalação de dependências: O projeto Iotivity possui dependência de algumas bibliotecas. Faça a instalação de acordo com a Figura 16.

$ sudo apt-get install libboost-dev libboost-program-options-dev libboost-dev uuid-dev libssl-dev libtool libglib2.0-dev libcap-dev libcurl14-opensssl-dev autotools-dev autoconf

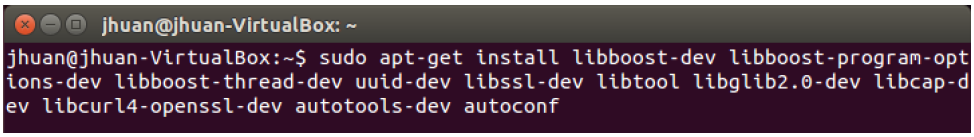


Figura 16 – Instalando algumas bibliotecas

5º Passo - Instalação do Boost C++: A biblioteca Boost C++ é necessária juntamente com a g++ para construção da pilha da Iotivity. Faça o *download* da bilblio-

teca boots c++ através deste link[3]. Após o *download* e dentro do diretório do *download*, realize o processo de descompactação, conforme explicitado na Figura 17.

$ sudo tar xzvf boost_1_55_0.tar.gz

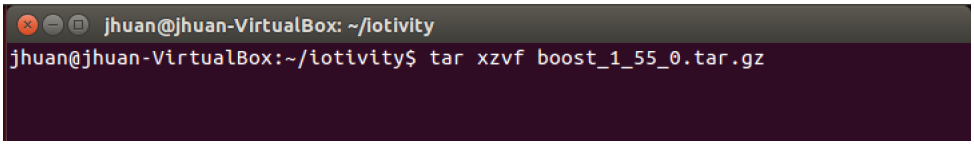


Figura 17 – Descompactando o download

No diretório que a biblioteca Boost C++ for descompactada, execute o comando descrito na Figura 18.

$ ./bootstrap.sh –with-libraries=system,filesystem,date_time,thread,regex, log, iostreams,program_options –prefix=/usr/local

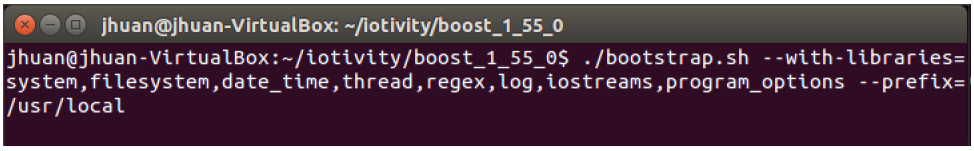


Figura 18 – Preparando para baixar as bibliotecas

Faça a instalação de bibliotecas adicionais, necessárias para o correto funcionamento do Boost C++. Note na Figura 19 os pacotes requeridos para instalação.

$ sudo apt-get install python-dev autotools-dev libicu-dev build-essential libbz2-dev

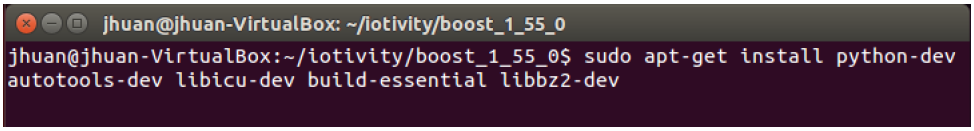


Figura 19 – Bibliotecas necessárias para o Boost C++

Note na Figura 20 o comando para início da instalação do Boost C++.

$ sudo ./b2 install

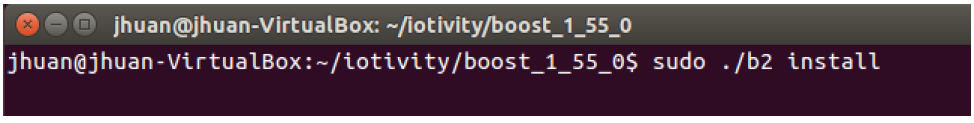


Figura 20 – Instalação do Boost C++

Para finalizar o processo de configuração, execute os comandos de acordo com a Figura 21.

[3] https://sourceforge.net/projects/boost/files/boost/1.55.0/boost_1_55_0.tar.gz/download

$ sudo sh -c 'echo '/usr/local/lib' » /etc/ld.so.conf.d/local.conf'

$ sudo ldconfig

```
jhuan@jhuan-VirtualBox: ~/iotivity/boost_1_55_0
jhuan@jhuan-VirtualBox:~/iotivity/boost_1_55_0$ sudo sh -c 'echo '/usr/local/lib
' >> /etc/ld.so.conf.d/local.conf'
```

Figura 21 – Processo de configuração do Boost C++

6º Passo - Instalação do Doxygen e SQlite: Doxygen é uma ferramenta para geração da documentação. Para instalá-la, utilize o comando no terminal conforme Figura 22.

$ sudo apt-get install doxygen

```
jhuan@jhuan-VirtualBox: ~/iotivity/boost_1_55_0
jhuan@jhuan-VirtualBox:~/iotivity/boost_1_55_0$ sudo apt-get install doxygen
```

Figura 22 – Instalação do Doxygen

Faça também a instalação das bibliotecas do SQLite, conforme explicitado na Figura 23.

$ sudo apt-get install sqlite3 libsqlite3-dev

```
jhuan@jhuan-VirtualBox: ~/iotivity
jhuan@jhuan-VirtualBox:~/iotivity/iotivity$ cd ..
jhuan@jhuan-VirtualBox:~/iotivity$ sudo apt-get install sqlite3 libsqlite3-dev
```

Figura 23 – Instalação das bibiotecas do SQlite

7º Passo - Configuração de autenticação por chaves: Crie uma conta na *The Linux Foundation* através deste link[4]. Esta conta será utilizada para os processos de autenticação no repositório da Iotivity. Em seguida, gere as chaves pública/privada que serão utilizadas para a conexão segura ao repositório da Iotivity.

$ ssh-keygen -t rsa -C "seu_email"

Após o processo de criação, acesse este o site do repositório[5], clique no nome de usuário e depois em *settings*. Clique no seu nome de usuário (Canto superior direito) e em seguida em *settings*. Clique em *"SSH Public Keys"* e em seguida *"Add key"*. Adicione a chave criada no 7º passo através da janela *"Add SSH Public Key"*.

8º Passo - Configuração do SSH: No diretório de configuração do SSH, crie um arquivo para armazenar as configurações de acesso. Adicione o seguinte conteúdo ao arquivo criado, conforme Figura 24:

[4] https://identity.linuxfoundation.org
[5] https://gerrit.iotivity.org/

Host iotivity gerrit.iotivity.org

IdentityFile ~ /.ssh/id_rsa

User [coloque o username da sua conta]

Port 29418

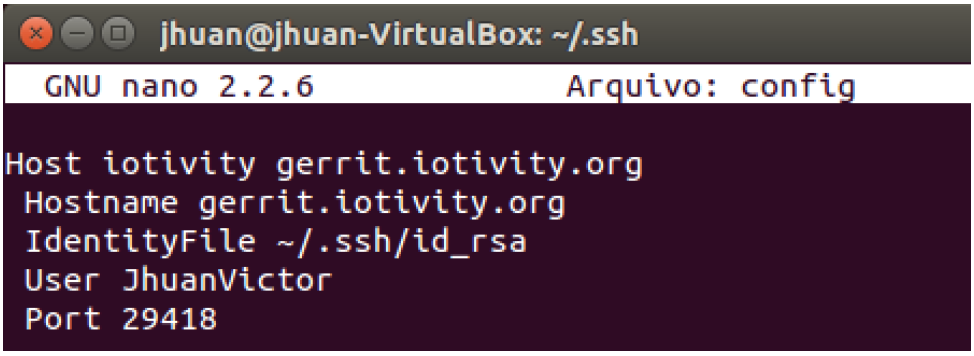


Figura 24 – Configurando SSH

Para verificar a conexão com o SSH, digite o comando:

$ ssh gerrit.iotivity.org

A mensagem "**** *Welcome do Gerrit Code Review* ****"é exibida caso a configuração seja realizada com sucesso. Note na Figura 25 a mensagem de boas-vindas recebida caso o usuário esteja com permissão de realizar o clone do repositório da Iotivity.

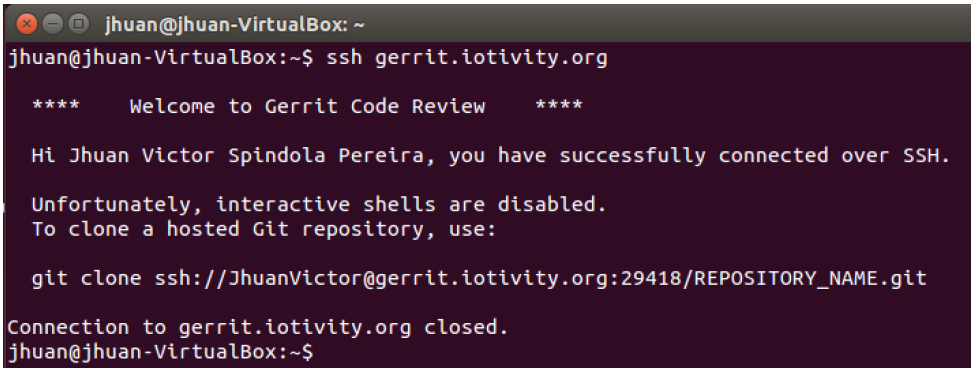


Figura 25 – Teste de conexão

9º Passo - Clone do repositório da Iotivity: Faça o clone do repositório através do comando no terminal:

$ git clone iotivity:iotivity

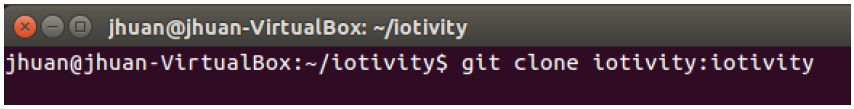


Figura 26 – Processo de clone do repositório do *framework*

Navegue ao diretório raiz do projeto Iotivity para construí-lo através do comando "scons"[6] (**$ scons**).Após o processo de compilação já é possível construir aplicações através do *Framework* Iotivity. Neste diretório[7] é possível encontrar exemplo de comunicação de objetos inteligentes, tais como a "SimpleServer"e "SimpleClient". A aplicação, em questão, simula a mudança do estado de um recurso compartilhado (lâmpada) entre objetos inteligentes, no qual o cliente é um observador do recurso compartilhado. O código do servidor realiza alterações na intensidade da luminosidade do recurso compartilhado e essas alterações são visualizadas pelo código cliente.

3.2.2 Microsoft Windows

Este tutorial irá ensinar a construir a Iotivity para o sistema operacional Microsoft Windows 10.

1º Passo - *Download* e Instalação do Visual Studio 2017: Faça o *download* do *software* Visual Studio 2017 através deste link[8]. Após o término do *download* e, após sua instalação, procure pelo recurso "Visual Studio Installer". Em Visual Studio Installer, clique na opção "Mais", conforme pode-se verificar na Figura 27.

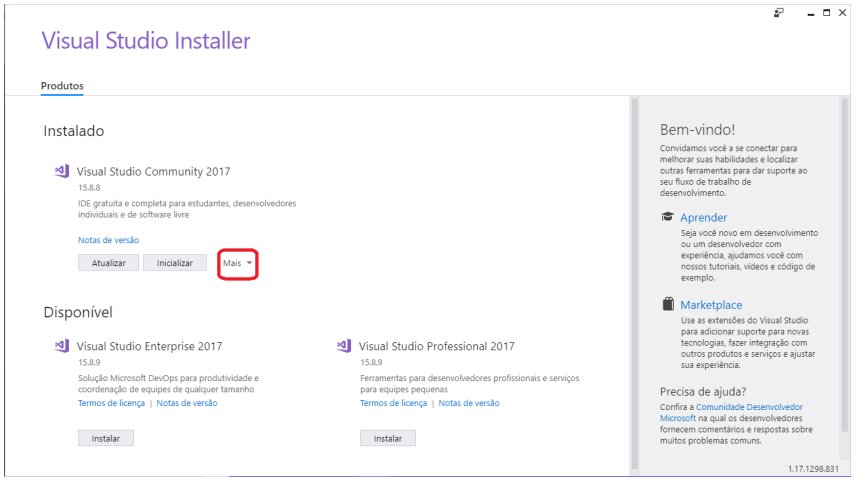


Figura 27 – Configuração do Visual Studio 2017 - Parte I

Realize a modificação na opção "Desenvolvimento para Desktop com C++"para a instalação dos componentes listados conforme Figura 28.

2º Passo - Instalação de Programas Adicionais: Durante a execução do scons, que ira compilar os códigos da Iotivity e realizar a construção do *framework*, será necessário a existência de programas adicionais. Note os programas adicionais necessários na Tabela 1:

6 Pode ser necessário o download de alguns pacotes adicionais, no momento do processo de compilação
7 iotivity/out/linux/x86_64/release/resource/examples
8 https://visualstudio.microsoft.com/pt-br/downloads/

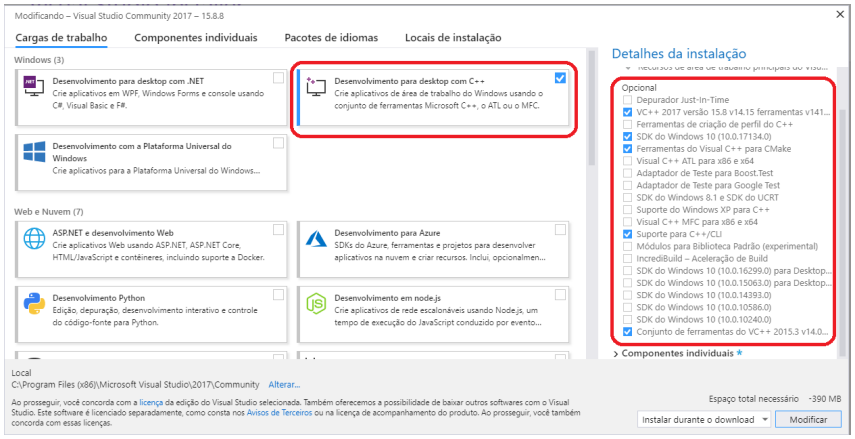


Figura 28 – Configuração do Visual Studio 2017 - Parte II

Programas Necessários	
Programa	**Versão**
Python	2.7.0
Scons	2.5.1
7-Zip	Mais atual
CMake	Mais atual

Tabela 1 – Programas necessários para a construção do framework Iotivity.

3º Passo - Configurações de variáveis de ambiente: É necessário configurar as variáveis de ambiente para os programas instalados no 2º passo. No Painel de Controle do Windows, selecione a opção "Sistema", e em seguida em "Configurações Avançadas do Sistema". Clique em "Variáveis de Ambiente". Selecione "*Path*"nas variáveis de sistema e clique em editar, conforme Figura 29. Adicione os seguintes caminhos, localizados na Tabela 2, nas variáveis de ambiente, conforme consta na Figura 30.

Variáveis de ambiente	
Programa	**Caminho**
Python	C:\Python27\
Python Scripts	C:\Python27\Scripts
CMake	C:\Program Files\CMake\bin
7-Zip	C:\Program Files\7-Zip

Tabela 2 – Caminhos dos programas que devem ser inseridos nas variáveis do sistema.

4º Passo - *Download* do *Framework*: O *download* do projeto será realizado através do git[9]. Após a instalação do git, crie uma pasta, em um local de sua preferência, abra um terminal e digite o comando para realizar o clone do *Framework Iotivity*:

git clone https://gerrit.iotivity.org/gerrit/iotivity

9 https://git-scm.com/downloads

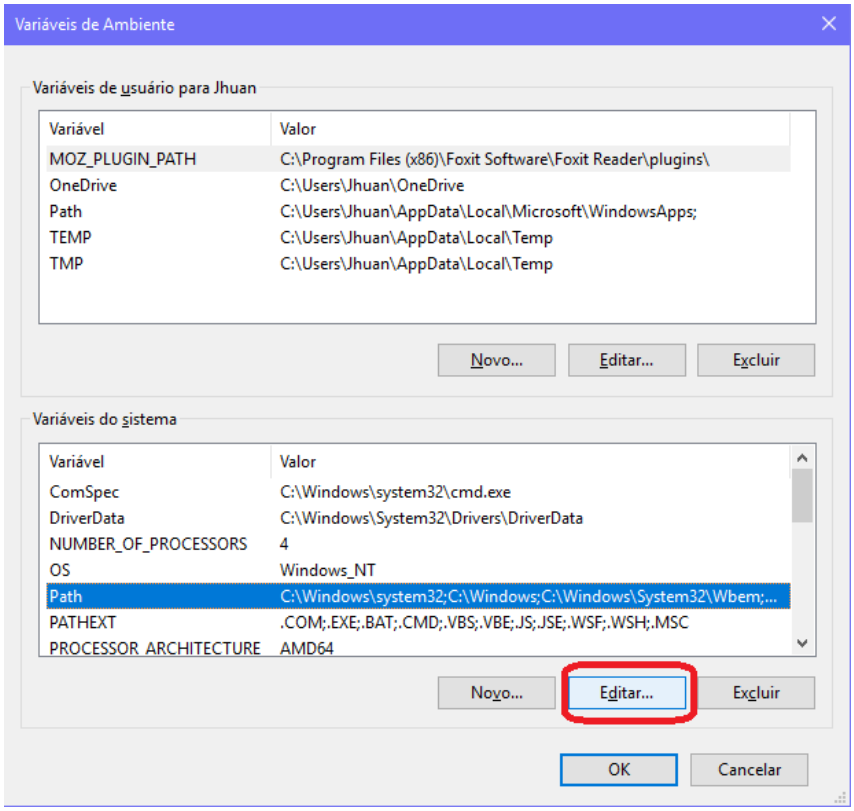


Figura 29 – Configurando as Variáveis de Ambiente

5º Passo - Configuração do *prompt* de comando do Visual Studio 2017: Para realizar a compilação do projeto é necessário abrir um terminal dentro do Visual Studio. Para que seja possível emular o terminal dentro do *software* Visual Studio, é necessário realizar uma configuração adicional[10] presente na seção "Executar o prompt de comando de dentro do Visual Studio".

6º Passo - Compilação do *Framework* Iotivity: A partir de um terminal de prompt de comando, dentro do Visual Studio, vá até o diretório do projeto da Iotivity, realizado no 4º passo, e faça:

$ run build

Durante o processo de compilação do *Framework* Iotivity no Windows 10, é possível que alguns erros ocorram. Segue lista de erros deparados e as soluções dos mesmos:

1º Erro - Falta de bibliotecas essenciais

No começo do processo de compilação, o scons pode alertar sobre a falta de algumas bibliotecas. Assim que o erro ocorrer, o próprio scons irá informar qual biblioteca esta faltando e mostrará um comando para clonar a biblioteca necessária. Após o clone da biblioteca necessária, execute o comando "run build"para reiniciar o processo de compilação.

[10] https://docs.microsoft.com/pt-br/dotnet/framework/tools/developer-command-prompt-for-vs

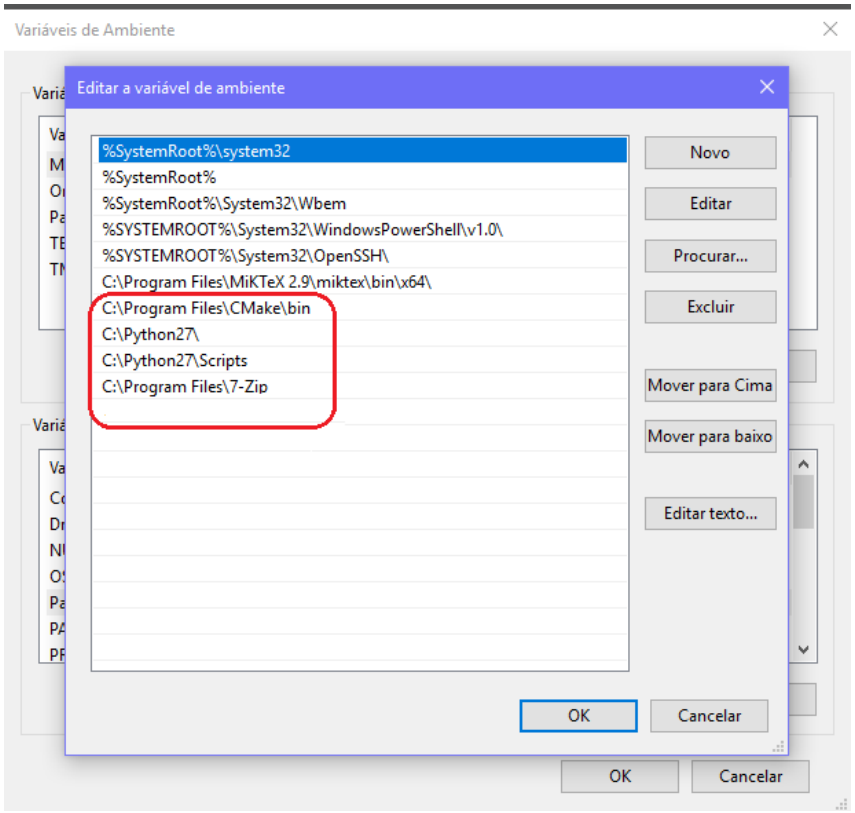


Figura 30 – Configurando as Variáveis de Ambiente

2º Erro - Falha no processo de descompactação automática de arquivos com extensão zip

Durante a compilação o scons realiza *download* de bibliotecas, componentes e programas externos, podendo ocorrer falhas durante a descompactação de arquivos com a extensão ".zip". No erro que aparecerá no *prompt* de comando, verifique o diretório em que o scons realiza o processo de descompactação de arquivos. Neste diretório, realize a descompactação manual do arquivo com extensão zip. Veja na Figura 31 um exemplo que mostra o referido erro ao descompactar o arquivo "boost_1_60_0.zip". O erro se resolve na com o processo de descompactação de arquivos feito manualmente.

3º Erro - Variável não inicializada corretamente

Caso seja exibida a mensagem "A variável stream não foi inicializada corretamente", altere o arquivo "external_builders.scons"pelo conteúdo do arquivo "erroExternalbuilders"que pode ser encontrado no seguinte repositório[11].

4º Erro - *Array index out of bounds* - cmd_list

Se ocorrer um erro no *array* "cmd_list", abra o arquivo SConscript, localizado no diretório iotivity\resource\csdk\security\provisioning\unittest\ e altere seu conteúdo pelo conteúdo do arquivo "erroSconscript"que pode ser encontrado neste diretório[12].

[11] https://github.com/JhuanVictor/Arquivos-para-corrigir-erros-da-Iotivity.git
[12] https://github.com/JhuanVictor/Arquivos-para-corrigir-erros-da-Iotivity.git

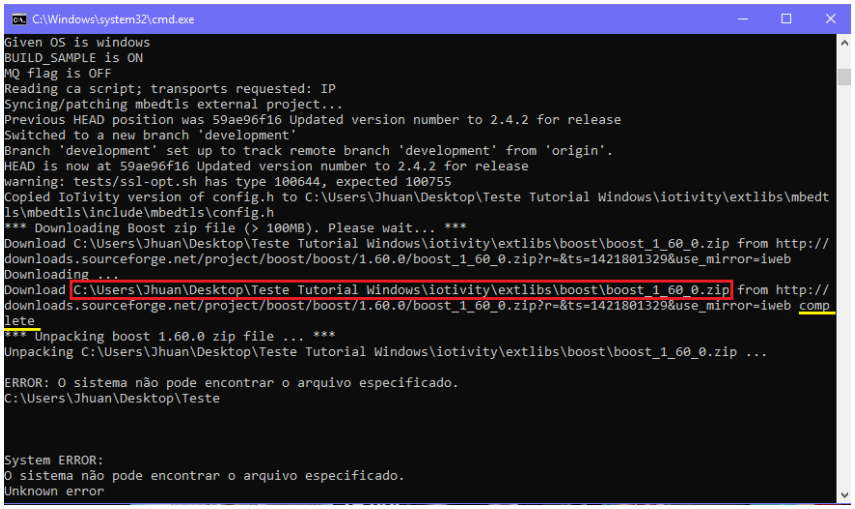


Figura 31 – Possíveis Erros - Descompactação Automática

3.3 Funções disponíveis no *Framework* Iotivity

3.3.1 Inicialização

A pilha de recursos da API possui duas arquiteturas de *software* multicamadas, uma para ambientes sem restrições computacionais, tais como Linux, Android e iOS, com APIs nas linguagens de programação C/C++ que possibilitam a conexão com dispositivos restritos e não restritos, por meio de Redes IP, e outra para dispositivos computacionais com restrições computacionais de processamento e memória. A API disponibilizada para dispositivos com limitações computacionais é apenas na linguagem C, devido questões de eficiência. Veja na Figura 32 a arquitetura básica e as camadas de *software* presentes no *framework* da Iotivity para dispositivos computacionais com diferentes capacidades computacionais.

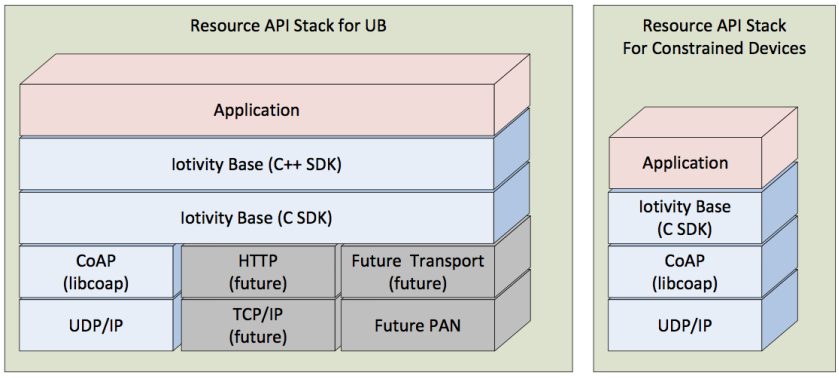


Figura 32 – Arquitetura Básica da IoTivity

A função *OCInit()* permite a inicialização da pilha, *OCProcess()* possibilita o processamento de baixo nível dos serviços da pilha e *OCStop()* função que possibilita que a pilha OC seja encerrada. Na API, disponível em C++, a função OCPlatform::Configure()

permite que um objeto seja substituído pela configuração de outro objeto padrão da OCPlatform. Veja na Figura 33 o fluxo de chamadas de funções realizada pela API em C++ quando é invocada.

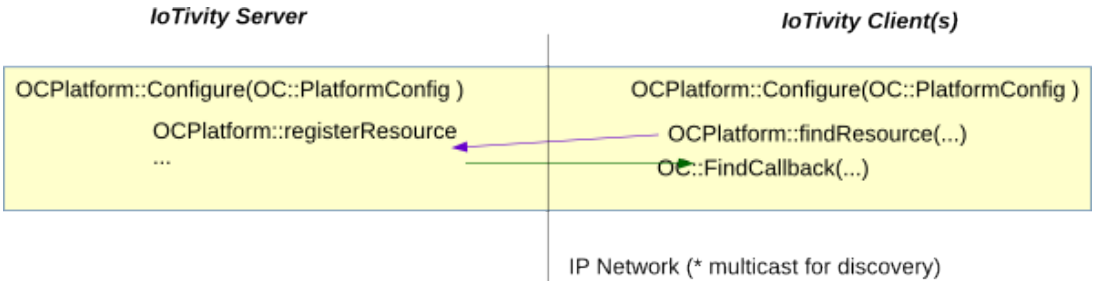


Figura 33 – Fluxo de chamadas da OCPlatform, API em C++

3.3.2 Registro e Descoberta de Recursos

A descoberta de recursos é uma camada extremamente importante na Iotivity. Essa camada consiste na descoberta e registro de recursos na rede. A funcionalidade de descoberta é uma operação cliente-servidor: um dispositivo servidor registra o recurso com a pilha, tornando os mesmos detectáveis por dispositivos que atuam como processos clientes. Os clientes descobrem recursos na rede por meio de solicitações de descoberta *multicast* ou *unicast*.

Para executar o registro de um recurso é necessário 2 (dois) itens: *Caminho URI*, um identificador para se alcançar o recurso, e um *Manipulador* que consiste em uma rotina usada para processar solicitações da pilha para o recurso.

3.3.3 Comunicação com os recursos

Este tópico trata sobre como é feita a comunicação entre recursos da Iotivity. A Figura 34 ilustra o fluxo de chamadas na comunicação entre dois recursos.

1. O aplicativo cliente invocada a função resource.get (. . .) para recuperar uma representação dos recursos.

2. A chamada, realizada no passo 1, é empacotada para a pilha que está sendo executada no processo ou fora do processo (*daemon*).

3. A API C é chamada para despachar a solicitação.

4. Nas aplicações em que o protocolo CoAP é utilizado para transporte, a camada inferior envia uma solicitação GET para o servidor de destino.

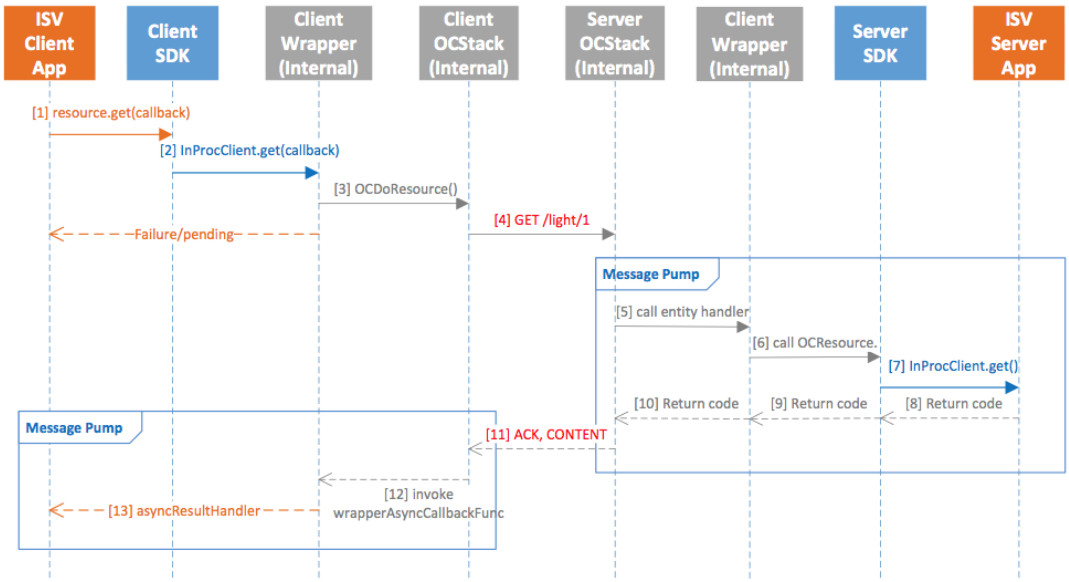


Figura 34 – Exemplo de comunicação entre recursos.

5. No lado do servidor, a função OCProcess () (message pump) recebe e analisa o pedido, em seguida despacha-o para o manipulador de entidade correto baseado no URI do pedido.

6. Nos casos em que a API C++ é utilizada, o manipulador de entidade C ++ analisa a carga útil e empacota-a para o aplicativo cliente, dependendo se a pilha do servidor está em execução no processo ou fora do processo (*daemon*).

7. O SDK do C ++ passa o manipulador C ++ associado ao OCResource.

8. O manipulador retorna o código de resultado e a representação para o SDK.

9. O SDK empacota o código de resultado e representação para o manipulador de entidade C ++.

10. O manipulador de entidade retorna o código e a representação do resultado para o protocolo CoAP.

11. O protocolo CoAP transporta os resultados para o dispositivo do cliente.

12. Os resultados retornam para chamada do OCDoResource.

13. Os resultados são retornados para a função asynResultCallback() do aplicativo cliente.

A função OCDoResource(), da API em C, é responsável pela descoberta de recursos ou executa solicitações em um recurso específico, por meio de uma URI. Já a função OCDoResponse() responde a uma determinada requisição.

3.3.4 Versionamento

Versionamento é a técnica utilizada para o problema do servidor e cliente estarem utilizando versões diferentes. A finalidade do versionamento é fazer os dois lados da comunicação conhecerem a versão que o outro utiliza. O versionamento abrange desde o nível de *framework* ao nível de aplicativo.

3.3.5 Observação de recursos

A observação de recursos possibilita a busca de recursos compartilhados disponíveis na rede e o registro de interesse em atuar como observador de um determinado recurso. Um observador será notificado sempre que um valor do recurso for alterado. A Figura 35 ilustra o fluxo de chamadas que são realizadas quando se pretende localizar e se registrar em um determinado recurso na rede.

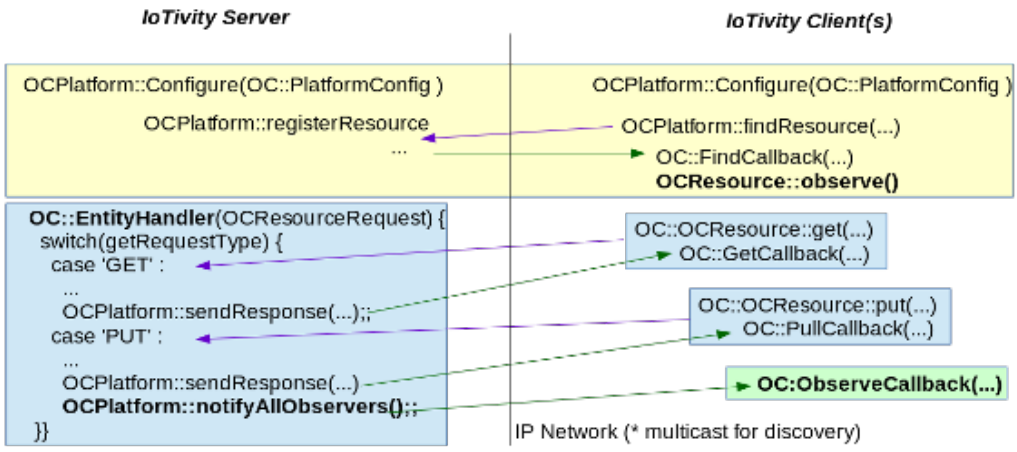


Figura 35 – Fluxo de chamadas para localizar e registrar-se como observador de um recurso.

3.3.6 Presença

Presença se refere à descoberta ativa ou de anúncio que acontece em nível de dispositivo. O servidor notifica sua presença para que o cliente possa se inscrever ou cancelar a assinatura dos eventos de presença de um dispositivo. O cliente hospeda o recurso que ativa a descoberta e o servidor é responsável pelas informações que são publicadas.

3.3.7 Recursos e suas propriedades

Recursos são objetos cadastrados na rede que possuem entidade física ou lógica. Todos recursos têm propriedades comuns que são especificadas nas respostas. Note na Tabela as propriedades comuns de um recurso:

Propriedades de um recurso	
Nome	Identificador legível.
Identidade	Identificador de instancia exclusivo.
Tipo	Categoria ou classe de um recurso.
Interface	Define como uma reposta será exibida.

Tabela 3 – Propriedades comuns de um recurso.

Nome: Forma legível do recurso, é um campo obrigatório.

Identidade: É um identificador de instancia exclusivo para um recurso. Esse valor pode ser um número ordinal exclusivo, uma string única ou um UUID.

Tipo: Essa propriedade define a categoria ou classe do recurso e é um campo obrigatório. Um recurso pode ter um ou mais tipos e são definidos na sua criação.

Interface: A interface possui duas utilidades, fornece uma visão sobre como uma resposta será exibida e detalhes de permissão para leitura e gravação para o recurso. Todos os recursos suportam uma interface de linha base (oic.if.baseline) e esta é uma interface padrão.

Interfaces que especificam como a resposta será retornada pelo servidor:

Baseline: Inclui todas as informações sobre o recurso, incluindo metadados e informações de coleta. É a interface padrão.

Linked List: Inclui apenas as informações de coleta sobre o recurso.

Batch: Permite a agregação de interação com todos os recursos. Cada recurso será interagido separadamente, mas suas respostas serão agregadas.

O outro tipo de interface está relacionado a permissões. Estes são relevantes para solicitações de recuperação e atualização.

Read: Permite que valores sejam lidos.

Read Write: Permite que valores sejam lidos e escritos.

Actuator: Permite criar, atualizar e recuperar valores do atuador.

Sensor: Permite que os valores do sensor sejam lidos.

3.3.8 Coleções

Coleções são contêineres que contêm referências a recursos. Para criar uma coleção, primeiro um recurso deve ser criado usando *createResource* e em seguida a API *bindResource* é usada para combinar recursos e formar uma coleção, a *UnBindResource* é

usada para retirar um recurso de uma coleção.

3.3.9 Gerenciador de provisionamento

O *Provisioning Manager* atua como o administrador de segurança dos dispositivos IoT em sua sub-rede IP. Assim que um dispositivo é inserido em uma sub-rede, o *Provisioning Manager* assume suas propriedades, fornecendo informações de segurança como por exemplo, credencial e política de controle de acesso, sendo possível gerenciar o novo dispositivo com segurança. Sem esse provisionamento, o novo aparelho recém introduzido na sub-rede poderia ser controlado por usuários indesejados que poderiam realizar ações que atrapalhem o funcionamento do dispositivo.

3.3.10 Gerenciador de segurança dos recursos

O SRM (*Security Resource Manager* - SRM) fornece o gerenciamento de recursos virtuais seguros (*Secure Virtual Resources* - SVRs) e controle de acesso com base em políticas definidas por SVRs. Os SVRs incluem recursos relacionados à segurança, como Listas de Controle de Acesso (*Access Control List* - ACLs), Estado do Proprietário do Dispositivo, Credenciais e Serviços de Segurança, eles são definidos em um banco de dados padrão e fornecidos ao SRM que é responsável por carregar, verificar e analisar esse banco de dados e mantê-lo na memória para atender a solicitações de decisão de acesso ou informações de credenciais.

Os SRM possuem duas funcionalidades principais, filtro de requisições e o manejo do SVR. No filtro de requisições o SRM recebe uma solicitação da camada abstração de conectividade e pode realizar três operações a partir desta, conceder o pedido, negar o pedido ou responder ao pedido do SVR. Ao realizar o manejo do SVR, o SRM gerencia o banco de dados de SVRs na memória ou no armazenamento persistente.

3.3.11 Encapsulamento de Recursos

Encapsulamento de recursos é uma camada abstrata que consiste em módulos de funções genéricas, comuns em todos os recursos. Fornece funções para o lado cliente e para o lado servidor, facilitando o trabalho dos desenvolvedores. No lado cliente, ele fornece funcionalidades como, cache de recursos e *broker*. O cache de recursos armazena em cache os dados do atributo de um recurso de interesse, essa funcionalidade consegue garantir isso utilizando um APIs de centros de dados privados, capazes de enviar/receber mensagens, *Getter/Setter* e cache de dados. A funcionalidade *broker* fornece as informações de acessibilidade dos recursos solicitados, além de também realizar a verificação de presença de recursos remotos. No servidor o encapsulamento traz uma maneira simples e direta

de criar um recurso e definir suas propriedades e atributos. Seu objetivo é basicamente oferecer módulos de funções comuns para facilitar a vida do desenvolvedor.

3.3.12 Contêiner de Recursos

O contêiner de recursos é um componente capaz de carregar dinamicamente as definições dos recursos configurados previamente. As definições são reunidas em pacotes configuráveis de recursos. Um caso de uso típico é fornecer traduções para protocolos externos e disponibilizar dispositivos ou serviços de terceiros como servidores de recursos da OIC.

3.3.13 Diretório de Recursos

O diretório de recursos é um diretório como serviço, onde contém informações sobre recursos que foram publicados na rede. Qualquer dispositivo pode atuar como um servidor de diretórios exceto dispositivos restritos. Armazena (cache) a identidade do recurso (endereço: porta) e propriedade (tipo de recurso, tipo de interface, etc) com suas informações de propriedade.

3.3.14 Easy Setup

Easy Setup é uma camada de serviço desenvolvida usando APIs nativas da plataforma Iotivity, para fazer dispositivos recém retirados da caixa serem facilmente conectados à rede Iotivity. O usuário pode transferir várias informações importantes para os novos dispositivos na fase de configuração da *Easy Setup*, que incluem: Informações de conexão *Wi-Fi AP* necessárias para o dispositivo se conectar ao *Home AP* e às configurações do dispositivo. Além disso, o usuário pode fornecer uma informação de acesso à nuvem para os dispositivos para que eles possam registrá-los em um servidor de nuvem IoTivity *(CoAP Native Cloud)* e o usuário possa acessá-los via nuvem IoTivity mesmo à distância. Na Figura 36, é possível observar um caso de uso do *Easy Setup*.

3.3.15 Gerenciador de Cenas

O gerenciador de cenas é implementado para ajudar os usuários a criarem uma cena com vários recursos descobertos em uma rede. O uso de cenas significa que os usuários podem alterar os recursos para uma representação configurada, ao executar uma cena, em vez de atualizar todos os recursos individualmente. O gerenciador de cena é desenvolvido em cima do Encapsulamento de recursos, um serviço primitivo Iotitivy. Na Figura 37 tem-se um exemplo desta funcionalidade, quando o usuário seleciona a cena "Assistindo Filme"em seu *smartphone* é enviada uma ordem ao *HUB* que desliga a luz e liga a caixa de som e a TV.

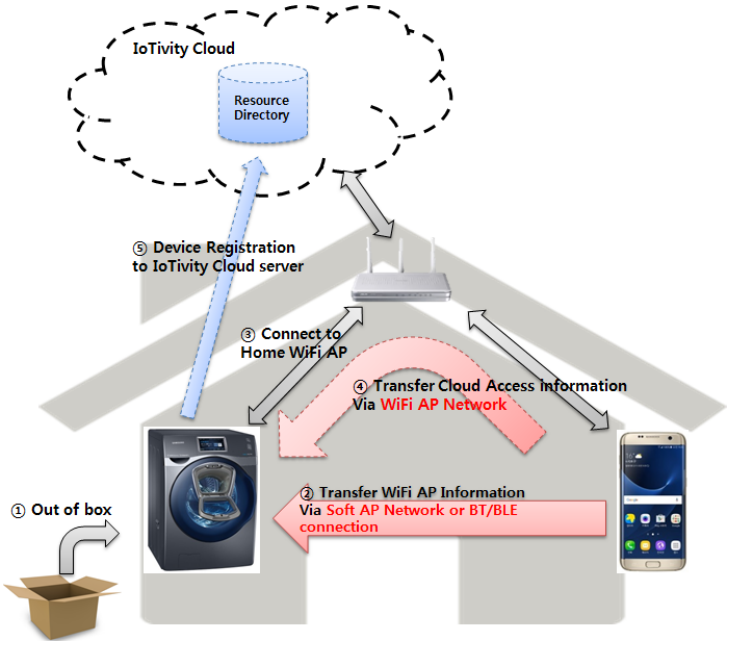


Figura 36 – Caso de uso da Easy Setup

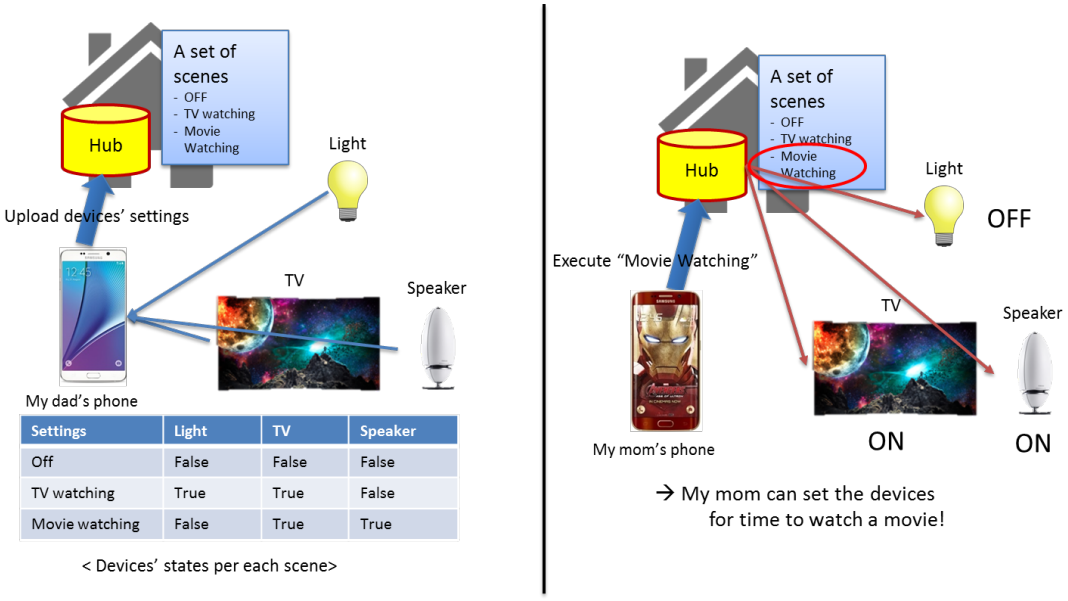


Figura 37 – Caso de uso do Gerenciador de Cenas

3.4 Considerações Finais

Neste capítulo apresentou-se detalhes da construção do *middleware* Iotivity para 2 (dois) de sistemas operacionais distintos: GNU/Linux Ubuntu e Microsoft Windows, além de trazer soluções para problemas percebidos durante o momento de compilação. Um explicação detalhada de cada funcionalidade do *middleware* também foi apresentada com o intuito de servir com base para a realizacão do estudo de caso apresentado no Capítulo 4.

4 Resultados

4.1 Introdução

Este capítulo inicia-se com a apresentação de uma aplicação, desenvolvida em C++, no qual servidor e cliente compartilham um recurso OCF. O recurso compartilhado, uma luz, é instanciada pelo servidor e tem seus atributos alterados de forma periódica. Os estados do recurso são observados pelo código cliente que recebe notificações das mudanças ocorridas. Em seguida, apresentamos uma aplicação de detecção e divulgação de alerta de erupção vulcânica, mostrando a possibilidade de comunicação, através do *framework* Iotivity, entre dispositivos inteligentes diferentes. A aplicação desenvolvida é testada em ambientes com sistemas operacionais distintos, o que prova sua capacidade de estabelecimento de conexão e troca de informações entre diferentes dispositivos computacionais.

4.2 Estudo de Caso 1: Compartilhamento de Recursos

O exemplo fornecido pela Iotivity simula um recurso OCF: uma luz. Este recurso fica no lado do servidor e possui duas propriedades: estado (*state*) e força (*power*). O cliente é capaz de atuar como observador do recurso e receber notificações sobre quaisquer mudanças que ocorrem no recurso compartilhado.

4.2.1 Código no Servidor

O servidor possui uma classe denominada LightResource, que possui 1 (um) construtor e 8 (oito) funções. O construtor define informações relacionadas ao recurso. Note na Tabela 4 as informações que o recurso possui:

Informações do recurso lâmpada	
Nome	Forma humana legível do recurso.
Estado	Lâmpada ligada ou desligada, True ou False.
Força	Intensidade da lâmpada.
URI	Caminho (path) para localização do recurso.
Handle	Uma rotina usada para processar solicitações para esse recurso.

Tabela 4 – Informações que são configuradas pelo construtor do recurso lâmpada.

A função createResource() é responsável pela criação do recurso, usando as informações definidas pelo construtor. Além disso, esta função define as propriedades do

recurso, definidas por flags. Um recurso definido com a flag OC_DISCOVERABLE pode ser descoberto pelos clientes que estão na mesma rede.

Para criar o recurso, a API registerResource é chamada e instancia todas informações do recurso. As funções, definidas na API registerResource, estão listadas na Tabela 5.

Funções da API registerResource	
getHandle()	retorna o handle do recurso.
put()	Recebe os novos valores e atualiza o estado interno do recurso.
post()	Cria um recurso na primeira execução.
get()	Atualiza a representação com o estado mais recente do recurso.
addType()	Adiciona o tipo ao handle do recurso.
addInterface()	Adiciona uma interface ao handle do recurso.
entityHandle()	Gerencia as requisições, atribuindo-as para a rotina correta.
ChangeLightRepresentation()	Altera a força da luz e notifica os observadores.

Tabela 5 – Funções presentas na API registerResource.

O recurso é então definido e setado como seguro (caso seja necessário) e se há a existência de uma lista pré-determinada de observadores. Ainda é realizada a definição do tipo de transporte, da qualidade de serviço suportada e o tipo de plataforma utilizada.

A classe LightResource é instanciada para criar o recurso, com a função createResource() e define o tipo e interface com as funçõe addType() e addInterface().

4.2.2 Código no Cliente

O objeto OCResourceIdentifier, combinado com a propriedade URI do OCResource, é usado para identificar recursos na rede. A classe Light serve como um recurso local que armazena os atributos que serão monitorados, a força e o estado da luz.

A função onObserve(), observa o recurso e mostra na tela as informações recebidas do servidor. Um contador é incrementado a cada mudança que ocorre no recurso: quando o número do contador chegar em 11 o recurso não é mais observado.

As funções onPost() e onPut() manipulam o retorno das chamadas de solicitação e com o auxílio das funções putLightRepresentation() e postLightRepresentation(), alteram o valor do recurso local. A função getLightRepresentation(), invoca a API get do recurso e envia o retorno para onGet(), que por sua vez formata a mensagem, de acordo com as informações que foram passadas no cabeçalho, e chama a função putLightRepresentation() para modificação do recurso localmente.

A função foundResource() utiliza o recurso compartilhado e obtém suas propriedades através esperando do recurso encontrando com URI igual a "/a/light". Quando o

recurso é encontrado, uma conexão com o servidor é estabelecida, através das informações obtidas por meio da função getServerHeaderOptions(). Após a conexão ser estabelecida, é possível obter informações do recurso e usar getLightRepresentation() para definir os valores do recurso instanciado localmente.

A plataforma no cliente é configurada através da especificação do tipo de transporte, da qualidade de serviço e do tipo de plataforma. Após a configuração da plataforma, a procura e descoberta automática de um recurso é iniciada.

Para complementar o estudo de caso, foi desenvolvida uma aplicação com base na Seção, aplicado ao problema de deteção e alerta de erupção vulcânica para demonstrar as funcionalidades do *framework*. A aplicação é desenvolvida para 2 (dois) sistemas operacionais diferentes (Windows 10 - Aplicação cliente e Ubuntu LTS 14.04 - Aplicação servidor), comprovando que é possível utilizar-se da Iotivity para estabelecimento da comunicação e troca de informações entre dispositivos heterogêneos.

4.3 Estudo de Caso 2: Sistema de Detecção e Divulgação de Alerta de Erupção Vulcânica Através do *Framework* Iotivity

A erupção de um vulcão pode resultar em um grave desastre natural, por vezes de consequências planetárias. Tal como outros eventos naturais, as erupções são imprevisíveis e causam danos indiscriminados.

Os vulcões fornecem alguns sinais anterior ao processo de erupção vulcânica. Estes sinais podem ser compreendidos como pequenos terremotos, inchaço, aumento na emissão de calor e de gases em suas aberturas e podem podem ser notados com antecedência, o que pode representar uma oportunidade de prevenção de possíveis desastres.

Essas alterações podem ser captadas através do monitorização sísmica, por satélites usados para detecção de possíveis inchaços no solo e por detectores terrestres (sensores) que medem as emissões de gás e calor.

Este estudo de caso tem como objetivo o desenvolvimento de uma aplicação que, aliada ao conceito de IoT juntamente com o *framework* da Iotivity, seja possível detectar o aumento de temperatura, decorrente de uma possível atividade vulcânica, e emitir um alerta para colaborar com a divulgação e a previsão de atividades vulcânicas.

De acordo com (SYMONDS, 1999) e o acontecimento ocorrido em 2018 no Havaí com o vulcão Kilauea que emitiu gases de lava vulcânica que atingiram a temperatura de 1.093 ºC (REVISTAVEJA, 2018), será considerada que a temperatura dos gases vulcânicos varie de 120 ºC a 1100 ºC. Deste modo, a aplicação desenvolvida funciona da seguinte forma: um dispositivo inteligente, juntamente com algum sensor de temperatura, são colocados próximos a borda de um vulcão, com o intuito de capturar alterações na

temperatura. Um segundo objeto inteligente, atuando como cliente, deve ser localizado em uma estação de emergência. Este atua como observador dos valores de temperatura obtidos pelo dispositivo inteligente localizado próximo/dentro do vulcão.

1. É colocado um dispositivo que possui como sistema operacional o Ubuntu LTS 14.04 juntamente com um sensor de temperatura que será responsável por medir a temperatura dos gases emitidos pelo vulcão antes de uma possível erupção. Neste dispositivo, a aplicação deve registrar o recurso do sensor na rede e gerenciar as assinaturas/requisições realizadas pelo cliente. através das funções disponíveis no *framework* Iotivity.

2. A aplicação cliente é executada na mesma rede porém em uma estação de emergência. Este dispositivo será notificado casa haja alerta de erupção. Considera-se que a aplicação cliente é executada em um dispositivo inteligente que possui o sistemas operacional Windows 10.

3. Quando executada a aplicação cliente é executa, a mesma irá procurar na rede por recursos que foram registrados pelo servidor localizado próximo a borda do vulcão, utilizando a busca de recursos presente nas funções da Iotivity. No momento em que o recurso é localizado, a aplicação irá fazer uma inscrição neste recurso para recebimento de informações na medida em que o sensor de temperatura captar uma modificação na valor lido/obtido. Note na Figura 38 a ilustração do sistema de detecção e divulgação de alerta de erupção vulcânica desenvolvido.

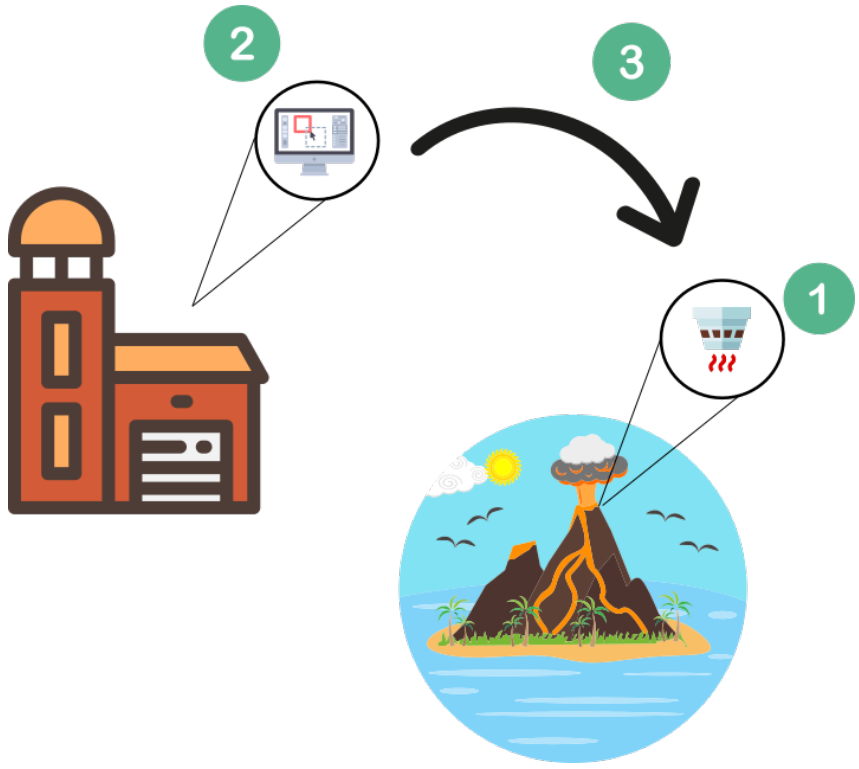


Figura 38 – Sistema de Detecção e Divulgação de Alerta de Erupção Vulcânica

4.4 Explicação detalhada da aplicação

Esta Seção contém uma explicação detalhada, a nível de código-fonte, sobre o funcionamento da aplicação de detecção e divulgação de alerta de erupção vulcânica através do uso do *framework* Iotivity.

4.4.1 Código no Servidor

O servidor da aplicação é responsável pela criação e registro do recurso na rede, para que observadores consigam localizá-lo e assim receber atualizações sempre que a temperatura for alterada. Um construtor instancia o recurso e essa nova instância é repassada para a função RegisterResource() da API C++, que registra o recurso no servidor. Veja na Seção 3.3.2 o modo de funcionamento de criação e registro de um recurso. Note a seguir o código-fonte que realiza o registro do recurso na rede.

```
1  void criarRecurso(){
2      //URI do recurso
3      std::string resourceURI = m_sensorUri;
4      //Tipo do Recurso. Neste caso, e temperatura
5      std::string resourceTypeName = "core.temperatura";
6      // Interface do recurso. Utilizaremos a interface padrao da
           iotivity.
7      std::string resourceInterface = DEFAULT_INTERFACE;
8
9      // OCResourceProperty e definido em ocstack.h
10     uint8_t resourceProperty;
11     if(isSecure){
12         resourceProperty = OC_DISCOVERABLE | OC_OBSERVABLE |
               OC_SECURE;
13     }
14     else{
15         resourceProperty = OC_DISCOVERABLE | OC_OBSERVABLE;
16     }
17     EntityHandler cb = std::bind(&RecursoTemperatura::
           entityHandler, this,PH::_1);
18
19     // Internamente sera criado e registrado o recurso
           temperatura.
20     OCStackResult result = OCPlatform::registerResource(
21                                m_resourceHandle, resourceURI,
                                       resourceTypeName,
```

```
22                                      resourceInterface, cb,
                                           resourceProperty);
23
24      if (OC_STACK_OK != result){
25          cout << "Criacao do recurso falhou!\n";
26      }
27  }
```

Após o registro do objeto, que neste caso é um sensor, o mesmo possuirá um URI completo (identificador para se alcançar o recurso), compostos da seguinte maneira: CoAP(s)+192.168.1.1:5246//Endereço-do-host/URI-especificada

O CoAP é um protocolo da camada de serviço, projetado para converter facilmente em HTTP para integração simplificada com a *Web*, destinado ao uso em dispositivos com recursos limitados, tais como em sistemas embarcados.

Com o recurso devidamente registrado, é realizado o manipulador de entidades (*Entity Handler*) que consiste em uma rotina usada para processar requisições feitas para a pilha do recurso. Este manipulador tem o objetivo processar requisições s *PUT, GET, POST e DELETE*, suprindo a necessidade da aplicação de sensoriamento da temperatura.

A função MudandoTemperatura(), responsável por simular a mudança de temperatura no vulcão, incrementa a variável que guarda a informação sobre a temperatura a cada 5 segundos e sempre que essa operação ocorre é enviada uma notificação para todos os observadores do recurso(Sensor). Simula-se aqui que a temperatura está alterando para um valor aleatório, porém poderia ser um valor lido através do sensor de temperatura acoplado ao dispositivo inteligente. Importante ressaltar que o processo de observação e a função MudandoTemperatura() são implementadas em *threads* para que possam ser executadas concorrencialmente. Observe o código da função que incrementa a variável temperatura, função MudandoTemperatura() e notifica os observadores.

```
1   void * MudandoTemperatura (void *param){
2       RecursoTemperatura* temperaturaPtr = (RecursoTemperatura*)
           param;
3
4       // Esta funcao monitora continuamente as mudancas
5       while (1){
6           sleep (5); //Tempo durante cada alteracao
7
8           if (gObservation){
9               // Se estiver sob observacao, quando houver alguma
                   alteracao no recurso de luz
10              // chamamos notificarObservadores.
11              // Para demonstracao, estamos mudando o valor da
```

```
            temperatura e notificando.
12          temperaturaPtr->m_temperatura += 35; // Muda a
               temperatura
13
14          cout << "\nNova Temperatura Detectada : " <<
               temperaturaPtr->m_temperatura << endl;
15          cout << "Notificando observadores com o Handle do
               recurso" << temperaturaPtr->getHandle() << endl;
16
17          OCStackResult result = OC_STACK_OK;
18          // Envia a notificacao para uma lista de observadores
                definida anteriormente
19          // ou para todos os observadores.
20          if(isListOfObservers){
21              std::shared_ptr<OCResourceResponse>
                   resourceResponse =
22                          {std::make_shared<OCResourceResponse
                               >()};
23
24              resourceResponse->setResourceRepresentation(
                   temperaturaPtr->get(), DEFAULT_INTERFACE);
25
26              result = OCPlatform::notifyListOfObservers(
                   temperaturaPtr->getHandle(),
27                          temperaturaPtr->
                               m_interestedObservers,
28                          resourceResponse);
29          }
30          else{
31              result = OCPlatform::notifyAllObservers(
                   temperaturaPtr->getHandle());
32          }
33
34          if(OC_STACK_NO_OBSERVERS == result){
35              cout << "Sem observadores, parando Notificacoes"
                   << endl;
36              gObservation = 0;
37          }
38      }
39  }
40
41  return NULL;
```

```
42 }
```

Desta forma, o fluxo da aplicação servidor é realizado da seguinte forma quando iniciado:

1. Criação e registro do recurso na rede.
2. Inicialização do manipulador de entidades.
3. Processo aguarda a assinatura de novos observadores com interesse em notificações sobre o recurso compartilhado.

No momento em que um cliente inicia o processo de observação:

1. O código servidor recebe a requisição de observação.
2. O manipulador de entidades realiza o processamento da requisição.

4.4.2 Código no Cliente

A aplicação no cliente tem como responsabilidade procurar e observar um recurso, enviando uma requisição para o servidor. A procura pelo recurso na rede é realizada através da função da API C++, findResource(). O objetivo é encontrar o recurso utilizando a URI definida pelo servidor,neste caso "/a/temperatura".

Assim que o recurso com a URI desejada é localizado, um registro para se tornar observador daquele recurso é realizado. Esta ação utiliza de uma funcionalidade do *framework, Resource Observation* que é explicada na seção 3.3.5.

Com o registro de observação concluido, a função onGet(), envia uma requisição do tipo *get* que solicita a versão mais recente das informações do recurso. Neste caso, a última temperatura captada pelo sensor. O servidor envia a reposta, e os valores são colocados em um recurso temporário que é passado para a função onObserve().

Na função onObserve() é encontrada a representação do recurso que armazena os dados mais recentes recebidos do servidor. Quando a função onGet() invoca a função onObserve(), a representação original é atualizada com as mesmas informações passadas pelo recurso temporário e é mostrado na tela a última temperatura que o sensor detectou. Seque a temperatura for maior do que 120 °C , valor mínimo de equilíbrio do gás vulcânico visto na seção 4.3, é exibida uma mensagem de alerta para uma possível erupção vulcânica. Note o código da função onObserve() que altera a representação do recurso com as novas informações recebidas do servidor.

```
1 void onObserve(const HeaderOptions /*Opcoes do cabecalho*/, const
      OCRepresentation& rep,
```

```
2                       const int& eCode, const int& sequenceNumber){
3       try{
4           if(eCode == OC_STACK_OK && sequenceNumber <=
                MAX_SEQUENCE_NUMBER){
5               if(sequenceNumber == OC_OBSERVE_REGISTER){
6                   std::cout << "Registro de observacao bem sucedida
                        !" << std::endl;
7               }
8
9               std::cout << "Resultado da Observacao:"<<std::endl;
10              rep.getValue("Estado", sensor.m_estado);
11              rep.getValue("Temperatura", sensor.m_temperatura);
12              rep.getValue("Nome", sensor.m_nome);
13
14              system("cls");
15              std::cout << "\tNome: " << sensor.m_nome << std::endl
                    ;
16              std::cout << "\tEstado: " << sensor.m_estado << std::
                    endl;
17              if (sensor.m_temperatura > 120){
18                  std::cout << "\n\t***Alerta*** A temperatura esta
                        aumentando! Perigo de erupcao vulcanica!!!" <<
                        std::endl;
19              }
20              std::cout << "\tTemperatura: " << sensor.
                    m_temperatura << std::endl;
21
22              if(observe_count() == 11){
23                  std::cout<<"Cancelando Observacao..."<<std::endl;
24                  OCStackResult result = curResource->cancelObserve
                        ();
25
26                  std::cout << "Resultado: "<< result <<std::endl;
27                  sleep(5);
28                  std::cout << "Cancelado com sucesso!"<<std::endl;
29                  std::exit(0);
30              }
31          }
32          else{
33              if(eCode == OC_STACK_OK){
34                  std::cout << "Cancelando..." << std::endl;
35              }
```

```
36              else{
37                  std::cout << "Reposta de erro onObserve: " <<
                        eCode << std::endl;
38                  std::exit(-1);
39              }
40          }
41      }
42      catch(std::exception& e){
43          std::cout << "Exception: " << e.what() << " Em onObserve"
                << std::endl;
44      }
45
46  }
```

O fluxo da aplicação cliente é realizado da seguinte forma quando iniciado (Após a execução da aplicação servidor):

1. Inicia-se o processo de localização do recurso na rede.
2. Solicita-se a última temperatura lida pelo sensor acoplado ao dispositivo inteligente.
3. Atualiza-se a representação local do recurso.
4. Mensagens são exibidas indicando a possibilidade de uma erupção vulcânica.

4.5 Considerações Finais

O *framework* Iotivity possibilita que diferentes objetos inteligentes comuniquem entre si através de suas funcionalidades. Neste Capítulo apresentou-se um estudo de caso, disponível no *framework*, no qual um recurso é compartilhado entre cliente e servidor. O servidor realiza alterações (mudança de força) no recurso (uma luz) e as alterações são percebidas pelo cliente. Em seguida, apresentamos uma aplicação de detecção e divulgação de alerta de erupção vulcânica: um dispositivo inteligente, juntamente com um sensor de temperatura, é colocado na borda de um vulcão para detecção da temperatura local. Caso a temperatura atinja limiar de alerta, emitimos um aviso a outro dispositivo inteligente localizado em uma estação de emergências. A natureza heterogêna dos dispositivos inteligentes, evidenciada pelo uso de diferentes sistemas operacionais, torna-se transparente devido ao uso do *framework* Iotivity para o estabelecimento dos processos de comunicação e troca de informações.

5 Considerações Finais e Trabalhos Futuros

5.1 Considerações finais

IoT possibilita que objetos do dia-a-dia se conectem à Internet, podendo ser controlados e acessados remotamente, e assim facilitando, agilizando e aprimorando diversas operações que realizamos no cotidiano. Para que se alcance todas essas vantagens é necessário solucionar alguns desafios de pesquisa. IoT permite que dispositivos de todos os tipos sejam conectados à Internet, o que implica em diferentes arquiteturas e capacidades computacionais.

Dado este ambiente heterogêneo, os processos de comunicação tornam-se tópico importante a ser discutido e analisado, alem de ser um fator de suma importância para que a IoT se expanda e se consolide. Iotivity é um projeto que tenta tratar esta dificuldade na comunicação dos objetos inteligentes, e foi escolhido como alvo deste estudo de caso por abordar e apresentar uma possível proposta para um problema de comunicação entre objetos inteligentes distintos presente em IoT e que pode contribuir muito para o avanço desta tecnologia.

O projeto da Iotivity possui duas API's, uma em C e outra em C++. As duas bibliotecas são bem vastas e com bastantes funcionalidades bem importantes, como por exemplo, descobrimento e registro de recursos, observação de recursos e presença em recursos. Um ponto positivo é que boa parte das funcionalidades possui uma versão para dispositivos robustos e outra para dispositivos restritos computacionalmente.

A Iotivity também disponibiliza uma *wiki* sobre seu *framework*. Nesta *wiki* é possível encontrar uma breve descrição sobre as funcionalidades do *framework* presente no tópico *Programming Guide*. Na seção *Getting Set up to Develop* possui uma explicação sobre todas as regras e ferramentas, que são utilizadas no projeto tais como, tutoriais de construção em diferentes plataformas e como propor mudanças no projeto. *Technical Notes* é outro tópico da *wiki* que explica o funcionamento do *framework*, como a Iotivity garante segurança, como funciona a comunicação, arquitetura, dentre outros.

A *wiki* possui bastante informações sobre o projeto, mas peca na forma de passar essas informações. O material disponibilizado não é muito didático o que pode acabar dificultando o aprofundamento de conhecimento sobre o *framework*. Outro ponto negativo é que as novas alterações demoram algum tempo para serem inseridas na *wiki*, por se tratar de um projeto colaborativo. Pode-se notar isso pelos tutoriais de construção: nos dois tutoriais utilizados neste trabalho (Windows e Linux). Os tutoriais de construção do *framework* apresentaram erros que foram tratados por este trabalho.

O objetivo deste TCC foi propor um material educacional livre, no qual seja possível aprofundar os conhecimentos sobre a IoT, como, definição, historia, arquitetura, funcionamento e aplicações. Confeccionar um material em português sobre o *framework* da Iotivity que auxilie novos membros a conhecerem e utilizarem este *framework*. Foi apresentado também m estudo de caso de detecção e divugalação de alerta de erupção vulcânica, a fim de explorar as principais funcionalidades do *framework* no contexto de estabelecimento e troca de informações entre dispositivos inteligentes heterogêneos.

5.2 Sugestão para Trabalhos Futuros

Com base nos resultados colhidos durante a confecção desta pesquisa exploratória sobre o *framework* da *Iotivity*, é sugerido os seguintes estímulos para a realização de trabalhos futuros:

- Desenvolver um método mais prático para construção do *framework Iotivity*.
- Desenvolver uma *wiki*, com os estudos de caso apresentados neste trabalho, para ser incorporada na documentação de *software* do *framework*.
- Desenvolver colaborativamente novas funcionalidades e propor melhorias, a nível de código-fonte, para o projeto *Iotivity*.

Referências

ALLIANCE, Z. *Zigbee IP and 920IP*. 2002. Disponível em: <https://www.zigbee.org/zigbee-for-developers/network-specifications/zigbeeip/>. Citado na página 19.

ALTAIR. *Carriots IoT*. 2017. Disponível em: <https://www.altairsmartworks.com/smartcore-overview>. Citado 2 vezes nas páginas 10 e 25.

CLOUD, G. *GOOGLE CLOUD IOT*. 2018. Disponível em: <https://cloud.google.com/solutions/iot/>. Citado 2 vezes nas páginas 10 e 24.

FARRELL, S. *Low-Power Wide Area Network (LPWAN) Overview*. 2018. Disponível em: <https://tools.ietf.org/html/rfc8376>. Citado na página 19.

FINEP. *Kevin Ashton – entrevista exclusiva com o criador do termo "Internet das Coisas"*. 2015. Disponível em: <http://finep.gov.br/noticias/todas-noticias/4446-kevin-ashton-entrevista-exclusiva-com-o-criador-do-termo-internet-das-coisas>. Citado na página 16.

FUTUREWEI, V. D. C. P. *Mobile Node Identifier Types for MIPv6*. 2018. Disponível em: <https://tools.ietf.org/html/rfc8371>. Citado 3 vezes nas páginas 16, 18 e 19.

GARTNERGROUP. *Hype cycle for emerging technologies*. 2018. Disponível em: <https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/>. Citado na página 13.

ICLINIC, B. *5 exemplos de IoT na área da saúde*. 2018. Disponível em: <https://blog.iclinic.com.br/exemplos-de-iot-na-area-da-saude/>. Citado na página 18.

IEEE. *EEE 802.15 WPAN Task Group 1 (TG1)*. 2018. Disponível em: <http://www.ieee802.org/15/pub/TG1.html>. Citado na página 19.

IETF. *INTERNET PROTOCOL*. 1981. Disponível em: <https://tools.ietf.org/html/rfc791>. Citado na página 18.

IETF. *Control and Provisioning of Wireless Access Points (CAPWAP) Protocol Binding for IEEE 802.11*. 2009. Disponível em: <https://tools.ietf.org/html/rfc5416>. Citado na página 19.

ISO, I. O. for S. *ISO/IEC 18092:2013 - Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)*. 2013. Disponível em: <https://www.iso.org/standard/56692.html>. Citado 2 vezes nas páginas 19 e 20.

JOIN, A. *Iotivity*. 2018. Disponível em: <https://iotivity.org>. Citado 2 vezes nas páginas 10 e 26.

KOCHE, J. C. *Fundamentos de metodologia científica*. [S.l.]: Editora Vozes, 2011. Citado na página 14.

KOODLI, R. *Mobile Networks Considerations for IPv6 Deployment*. 2011. Disponível em: <https://tools.ietf.org/html/rfc6342>. Citado na página 19.

LINKSMART. *Create your internet of things*. 2018. Disponível em: <https://www.linksmart.eu>. Citado 2 vezes nas páginas 10 e 25.

NEGOCIOS/GLOBO, R. P. empresas grandes. *COMO A INTERNET DAS COISAS É USADA NO BRASIL*. 2017. Disponível em: <https://revistapegn.globo.com/Tecnologia/noticia/2017/10/3-exemplos-de-como-internet-das-coisas-ja-e-usada-no-brasil.html>. Citado 2 vezes nas páginas 17 e 18.

NGUYEN MARYLINE LAURENT B, N. O. K. T. Survey on secure communication protocols for the internet of things. *Communicating Systems Laboratory, 91191 Gif-sur-Yvette CEDEX, France*, 2015. Citado na página 13.

OPENIOT. *Project OpenIoT*. 2018. Disponível em: <http://www.openiot.eu>. Citado 2 vezes nas páginas 10 e 26.

PIRES, P. de F. *Proposta para Grupo de Trabalho - GT-EcoDiF: Ecossistema Web de Dispositivos Físicos*. 2012. Disponível em: <https://memoria.rnp.br/_arquivo/gt/2012/GT_EcoDif.pdf>. Citado 2 vezes nas páginas 10 e 24.

PLUMMER, D. C. *An Ethernet Address Resolution Protocol*. 1982. Disponível em: <https://tools.ietf.org/html/rfc826>. Citado na página 19.

REVISTAVEJA. *Gases de lava vulcânica criam nova ameaça no Havaí*. 2018. Disponível em: <https://veja.abril.com.br/mundo/gases-de-lava-vulcanica-criam-nova-ameaca-no-havai/>. Citado na página 47.

SANTOS, B. P. et al. Internet das coisas: da teoriaa prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, 2016. Citado 3 vezes nas páginas 13, 18 e 21.

SWARTZ, A. *application/rdf+xml Media Type Registration*. 2004. Disponível em: <https://www.ietf.org/rfc/rfc3870.txt>. Citado na página 21.

SYMONDS, R. Gases, volcanic. In: _____. *Environmental Geology*. Dordrecht: Springer Netherlands, 1999. p. 270–271. ISBN 978-1-4020-4494-6. Disponível em: <https://doi.org/10.1007/1-4020-4494-1_146>. Citado na página 47.

TANENBAUM, A. S. *Computer Networks*. [S.l.]: Pearson, 2002. Citado na página 13.

THOMSEN, A. *O que é Arduino?* 2014. Disponível em: <https://www.filipeflop.com/blog/o-que-e-arduino/>. Citado na página 23.

W3C. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. 2012. Disponível em: <https://www.w3.org/TR/owl2-syntax/>. Citado na página 21.

W3C. *Efficient Extensible Interchange Working Group Public Page*. 2016. Disponível em: <https://www.w3.org/XML/EXI/>. Citado na página 21.

WEISER, M. The computer for the 21st century. *Mobile Computing and Communications Review*, v. 3, n. 3, p. 3–11, 1999. Citado na página 16.

Apêndices

APÊNDICE A – Código Fonte

A.1 Código Servidor

```
1
2  #include "iotivity_config.h"
3
4  #include <functional>
5  #ifdef HAVE_UNISTD_H
6  #include <unistd.h>
7  #endif
8
9  #ifdef HAVE_PTHREAD_H
10 #include <pthread.h>
11 #endif
12
13 #include <mutex>
14 #include <condition_variable>
15 #include "OCPlatform.h"
16 #include "OCApi.h"
17
18 #ifdef HAVE_WINDOWS_H
19 #include <windows.h>
20 #endif
21
22 #include "ocpayload.h"
23
24 #include "stdlib.h"
25
26 using namespace OC;
27 using namespace std;
28 namespace PH = std::placeholders;
29
30 static const char* SVR_DB_FILE_NAME = "./oic_svr_db_server.dat";
31 int gObservation = 0;
32 void * MudandoTemperatura (void *param);
33 void * handleSlowResponse (void *param, std::shared_ptr<
      OCResourceRequest> pRequest);
34
```

```
35 // Configurando informacoes da plataforma
36 std::string gPlatformId = "0A3E0D6F-DBF5-404E-8719-D6880042463A";
37 std::string gManufacturerName = "OCF";
38 std::string gManufacturerLink = "https://www.iotivity.org";
39 std::string gModelNumber = "myModelNumber";
40 std::string gDateOfManufacture = "2018-11-13";
41 std::string gPlatformVersion = "myPlatformVersion";
42 std::string gOperatingSystemVersion = "myOS";
43 std::string gHardwareVersion = "myHardwareVersion";
44 std::string gFirmwareVersion = "1.0";
45 std::string gSupportLink = "https://www.iotivity.org";
46 std::string gSystemTime = "2016-01-15T11.01";
47
48 // Configurando campos de informacoes do dispositivo
49 std::string  deviceName = "Servidor da Aplicacao do Vulcao";
50 std::string  deviceType = "oic.wk.tv";
51 std::string  specVersion = "ocf.1.1.0";
52 std::vector<std::string> dataModelVersions = {"ocf.res.1.1.0", "
      ocf.sh.1.1.0"};
53 std::string  protocolIndependentID = "fa008167-3bbf-4c9d-8604-
      c9bcb96cb712";
54
55
56 // OCPlatformInfo Contem todas as informacoes da plataforma
57 OCPlatformInfo platformInfo;
58
59 // Especifica se serao notificados todos os observadores ou
      somente uma determinada lista de observadores.
60 // false: Notifica todos os observadores.
61 // true: Notifica uma determinada lista de observadores.
62 bool isListOfObservers = false;
63
64 // Especifica se o recurso estara seguro ou nao seguro
65 // como esta aplicacao e apenas umas simulacao nao se torna
      necessario utilizar a opcao segura.
66 // false: Recurso nao seguro.
67 // true: Recurso seguro.
68 bool isSecure = false;
69
70 /// Specifies whether Entity handler is going to do slow response
       or not
71 bool isSlowResponse = false;
```

```
72
73 /// Esta classe representa um unico recurso chamado '
      RecursoTemperatura'. Este recurso tem
74 /// duas propriedades simples denominadas 'estado' e 'poder'
75
76 class RecursoTemperatura{
77
78 public:
79     std::string m_nome;
80     bool m_estado;
81     int m_temperatura;
82     std::string m_lightUri;
83     OCResourceHandle m_resourceHandle;
84     OCRepresentation m_lightRep; //Nao pode ser alterado.
85     ObservationIds m_interestedObservers;
86
87     public:
88     /// Construtor
89     RecursoTemperatura()
90         :m_nome("Sensor de Temperatura"), m_estado(false),
              m_temperatura(0), m_lightUri("/a/temperatura"), //teste
91         m_resourceHandle(nullptr) {
92         // Inicializa a representacao do recurso.
93         m_lightRep.setUri(m_lightUri);
94
95         m_lightRep.setValue("Estado", m_estado);
96         m_lightRep.setValue("Temperatura", m_temperatura);
97         m_lightRep.setValue("Nome", m_nome);
98     }
99
100         void criarRecurso(){
101             //URI do recurso
102             std::string resourceURI = m_lightUri;
103             //Tipo do Recurso. Neste caso, e temperatura
104             std::string resourceTypeName = "core.temperatura";
105             // Interface do recurso. Utilizaremos a interface
                   padrao da iotivity.
106             std::string resourceInterface = DEFAULT_INTERFACE;
107
108             // OCResourceProperty e definido em ocstack.h
109             uint8_t resourceProperty;
110             if(isSecure){
```

```
111                 resourceProperty = OC_DISCOVERABLE |
                        OC_OBSERVABLE | OC_SECURE;
112             }
113             else{
114                 resourceProperty = OC_DISCOVERABLE |
                        OC_OBSERVABLE;
115             }
116             EntityHandler cb = std::bind(&RecursoTemperatura::
                    entityHandler, this,PH::_1);
117
118             // Internamente sera criado e registrado o recurso
                    temperatura.
119             OCStackResult result = OCPlatform::registerResource(
120                                         m_resourceHandle,
                                                resourceURI,
                                                resourceTypeName,
121                                         resourceInterface, cb,
                                                resourceProperty);
122
123             if (OC_STACK_OK != result){
124                 cout << "Criacao do recurso falhou!\n";
125             }
126         }
127
128         OCStackResult criarRecurso1(){
129             //URI do recurso
130             std::string resourceURI = "/a/light1";
131             //Tipo do Recurso. Neste caso, e temperatura
132             std::string resourceTypeName = "core.temperatura";
133             // Interface do recurso. Utilizaremos a interface
                    padrao da iotivity.
134             std::string resourceInterface = DEFAULT_INTERFACE;
135
136             // OCResourceProperty e definido em ocstack.h
137             uint8_t resourceProperty;
138             if(isSecure){
139                 resourceProperty = OC_DISCOVERABLE |
                        OC_OBSERVABLE | OC_SECURE;
140             }
141             else{
142                 resourceProperty = OC_DISCOVERABLE |
                        OC_OBSERVABLE;
```

```
143             }
144             EntityHandler cb = std::bind(&RecursoTemperatura::
                    entityHandler, this,PH::_1);
145
146             OCResourceHandle resHandle;
147
148             // Internamente sera criado e registrado o recurso
                    temperatura.
149             OCStackResult result = OCPlatform::registerResource(
150                                         resHandle, resourceURI,
                                            resourceTypeName,
151                                         resourceInterface, cb,
                                            resourceProperty);
152
153             if (OC_STACK_OK != result){
154                 cout << "Criacao do recurso falhou!\n";
155             }
156
157             return result;
158         }
159
160
161         OCResourceHandle getHandle(){
162             return m_resourceHandle;
163         }
164
165         // Coloca a representacao.
166         // Pega os valores da representacao e
167         // atualiza o estado interno.
168         void put(OCRepresentation& rep){
169             try {
170                 if (rep.getValue("Estado", m_estado)){
171                     cout << "\t\t\t\t" << "Estado: " << m_estado
                            << endl;
172                 }
173                 else{
174                     cout << "\t\t\t\t" << "Estado nao encontrado
                            na representacao" << endl;
175                 }
176
177                 if (rep.getValue("Temperatura", m_temperatura)){
178                     cout << "\t\t\t\t" << "Temperatura: " <<
```

```
                        m_temperatura << endl;
179                 }
180                 else{
181                     cout << "\t\t\t\t" << "Temperatura nao
                            encontrada na representacao" << endl;
182                 }
183             }
184             catch (exception& e){
185                 cout << e.what() << endl;
186             }
187
188         }
189
190         // Post representacao.
191         // Post pode criar novo recurso ou simplesmente agir como
                put.
192         // Obtem valores da representacao e
193         // atualiza o estado interno.
194         OCRepresentation post(OCRepresentation& rep){
195             static int first = 1;
196
197             // pela primeira vez, tenta criar um recurso.
198             if(first){
199                 first = 0;
200
201                 if(OC_STACK_OK == criarRecurso1()){
202                     OCRepresentation rep1;
203                     rep1.setValue("createduri", std::string("/a/
                            temperatura1"));
204
205                     return rep1;
206                 }
207             }
208
209             // Da segunda vez em diante apenas realiza o put.
210             put(rep);
211             return get();
212         }
213
214
215         // obtem a representacao atualizada.
216         // Atualiza a representacao com o estado interno mais
```

```
            recente antes
217         // de enviar.
218         OCRepresentation get(){
219             m_lightRep.setValue("Estado", m_estado);
220             m_lightRep.setValue("Temperatura", m_temperatura);
221 
222             return m_lightRep;
223         }
224 
225         void addType(const std::string& type) const{
226             OCStackResult result = OCPlatform::bindTypeToResource
                    (m_resourceHandle, type);
227             if (OC_STACK_OK != result){
228                 cout << "Falha ao vincular TypeName ao recurso!\n
                        ";
229             }
230         }
231 
232         void addInterface(const std::string& iface) const{
233             OCStackResult result = OCPlatform::
                    bindInterfaceToResource(m_resourceHandle, iface);
234             if (OC_STACK_OK != result){
235                 cout << "Falha ao vincular Interface ao recurso!\
                        n";
236             }
237         }
238 
239     private:
240     // Esta e apenas uma implementacao de amostra do manipulador
            de entidade.
241     // O manipulador de entidades pode ser implementado de varias
             maneiras pelo fabricante.
242     OCEntityHandlerResult entityHandler(std::shared_ptr<
            OCResourceRequest> request){
243         cout << "\tNo manipulador de entidade CPP do servidor:\n"
                ;
244         OCEntityHandlerResult ehResult = OC_EH_ERROR;
245         if(request){
246             // Obtem o tipo de requisicao e as flags de
                    requisicao
247             std::string requestType = request->getRequestType();
248             int requestFlag = request->getRequestHandlerFlag();
```

```
249
250             if(requestFlag & RequestHandlerFlag::RequestFlag){
251                 cout << "\t\trequestFlag : Requisicao\n";
252                 auto pResponse = std::make_shared<OC::
                        OCResourceResponse>();
253                 pResponse->setRequestHandle(request->
                        getRequestHandle());
254                 pResponse->setResourceHandle(request->
                        getResourceHandle());
255
256                 // Verificando se ha parametros de consulta (se
                        houver)
257                 QueryParamsMap queries = request->
                        getQueryParameters();
258
259                 if (!queries.empty()){
260                     std::cout << "\nProcessamento de consultas
                            ate entityHandler" << std::endl;
261                 }
262                 for (auto it : queries){
263                     std::cout << "Query key: " << it.first << "
                            value : " << it.second
264                             << std:: endl;
265                 }
266
267                 // Se o tipo de requisicao for GET
268                 if(requestType == "GET"){
269                     cout << "\t\t\tTipo de Requisicao : GET\n";
270                     if(isSlowResponse){ // Reposta Lenta
271                         static int startedThread = 0;
272                         if(!startedThread)
273                         {
274                             std::thread t(handleSlowResponse, (
                                    void *)this, request);
275                             startedThread = 1;
276                             t.detach();
277                         }
278                         ehResult = OC_EH_SLOW;
279                     }
280                     else{ // Resposta normal.
281
282                         pResponse->setResponseResult(OC_EH_OK);
```

```
283                         pResponse->setResourceRepresentation(get
                               ());
284                         if(OC_STACK_OK == OCPlatform::
                               sendResponse(pResponse)){
285                             ehResult = OC_EH_OK;
286                         }
287                     }
288                 }
289                 else if(requestType == "PUT"){
290                     cout << "\t\t\tTipo de Requisicao : PUT\n";
291                     OCRepresentation rep = request->
                           getResourceRepresentation();
292
293
294                     // Faz operacoes relacionadas a requisicao
                           PUT
295                     // Atualiza o RecursoTemperatura
296                     put(rep);
297
298                     pResponse->setResponseResult(OC_EH_OK);
299                     pResponse->setResourceRepresentation(get());
300                     if(OC_STACK_OK == OCPlatform::sendResponse(
                           pResponse)){
301                         ehResult = OC_EH_OK;
302                     }
303                 }
304                 else if(requestType == "POST"){
305                     cout << "\t\t\trequestType : POST\n";
306
307                     OCRepresentation rep = request->
                           getResourceRepresentation();
308
309                     // Faz operacoes relacionadas a requisicoes
                           POST
310                     OCRepresentation rep_post = post(rep);
311                     pResponse->setResourceRepresentation(rep_post
                           );
312
313                     if(rep_post.hasAttribute("createduri")){
314                         pResponse->setResponseResult(
                               OC_EH_RESOURCE_CREATED);
315                         pResponse->setNewResourceUri(rep_post.
```

```
                            getValue<std::string>("createduri"));
316                 }
317                 else{
318                     pResponse->setResponseResult(OC_EH_OK);
319                 }
320
321                 if(OC_STACK_OK == OCPlatform::sendResponse(
                       pResponse)){
322                     ehResult = OC_EH_OK;
323                 }
324             }
325             else if(requestType == "DELETE"){
326                 cout << "Delete request received" << endl;
327             }
328         }
329
330         if(requestFlag & RequestHandlerFlag::ObserverFlag){
331             ObservationInfo observationInfo = request->
                   getObservationInfo();
332             if(ObserveAction::ObserveRegister ==
                   observationInfo.action){
333                 m_interestedObservers.push_back(
                       observationInfo.obsId);
334             }
335             else if(ObserveAction::ObserveUnregister ==
                   observationInfo.action){
336                 m_interestedObservers.erase(std::remove(
337                                                         m_interest
                                                             .
                                                             begin
                                                             ()
                                                             ,
338                                                         m_interest
                                                             .
                                                             end
                                                             ()
                                                             ,
339                                                         observatio
                                                             .
                                                             obsId
```

```
                                                                    )
                                                                    ,
340                                                             m_interest
                                                                    .
                                                                    end
                                                                    ()
                                                                    )
                                                                    ;
341             }
342
343
344             #if defined(_WIN32)
345             DWORD threadId = 0;
346             HANDLE threadHandle = INVALID_HANDLE_VALUE;
347             #else
348                         pthread_t threadId;
349             #endif
350
351             cout << "\t\trequestFlag : Observer\n";
352             gObservation = 1;
353             static int startedThread = 0;
354
355             // A observacao acontece em uma thread diferente
                   da funcao MudandoTemperatura.
356             if(!startedThread){
357                 #if defined(_WIN32)
358                     threadHandle = CreateThread(NULL, 0, (
                           LPTHREAD_START_ROUTINE)
                           MudandoTemperatura, (void*)this, 0, &
                           threadId);
359                 #else
360                     pthread_create (&threadId, NULL,
                           MudarTemperaturaRepresentacao, (void *)
                           this);
361                 #endif
362                 startedThread = 1;
363             }
364             ehResult = OC_EH_OK;
365         }
366     }
```

```
367         else{
368             std::cout << "Requisicao invalida" << std::endl;
369         }
370         return ehResult;
371     }
372 };
373
374
375 // MudarTemperaturaRepresentacao e uma funcao de observacao,
376 // que notifica qualquer alteracao no recurso para a pilha
377 // via notifyObservers.
378
379 void * MudandoTemperatura (void *param){
380     RecursoTemperatura* temperaturaPtr = (RecursoTemperatura*)
        param;
381
382     // Esta funcao monitora continuamente as mudancas
383     while (1){
384         sleep (5); //Tempo durante cada alteracao
385
386         if (gObservation){
387             // Se estiver sob observacao, quando houver alguma
                alteracao no recurso de luz
388             // chamamos notificarObservadores.
389             // Para demonstracao, estamos mudando o valor da
                potencia e notificando.
390             temperaturaPtr->m_temperatura += 35; // Muda a
                temperatura
391
392             cout << "\nNova Temperatura Detectada : " <<
                temperaturaPtr->m_temperatura << endl;
393             cout << "Notificando observadores com o Handle do
                recurso" << temperaturaPtr->getHandle() << endl;
394
395             OCStackResult result = OC_STACK_OK;
396
397             if(isListOfObservers){
398                 std::shared_ptr<OCResourceResponse>
                    resourceResponse =
399                             {std::make_shared<OCResourceResponse
                                >()};
400
```

```
401                 resourceResponse->setResourceRepresentation(
                       temperaturaPtr->get(), DEFAULT_INTERFACE);
402
403                 result = OCPlatform::notifyListOfObservers(
                       temperaturaPtr->getHandle(),
404                                                                 temperaturaPtr
                                                                    ->
                                                                    m_interest
                                                                    ,
405                                                                 resourceRespo
                                                                    );
406             }
407             else{
408                 result = OCPlatform::notifyAllObservers(
                       temperaturaPtr->getHandle());
409             }
410
411             if(OC_STACK_NO_OBSERVERS == result){
412                 cout << "Sem observadores, parando Notificacoes"
                       << endl;
413                 gObservation = 0;
414             }
415         }
416     }
417
418     return NULL;
419 }
420
421 void DeletePlatformInfo(){
422     delete[] platformInfo.platformID;
423     delete[] platformInfo.manufacturerName;
424     delete[] platformInfo.manufacturerUrl;
425     delete[] platformInfo.modelNumber;
426     delete[] platformInfo.dateOfManufacture;
427     delete[] platformInfo.platformVersion;
428     delete[] platformInfo.operatingSystemVersion;
429     delete[] platformInfo.hardwareVersion;
430     delete[] platformInfo.firmwareVersion;
431     delete[] platformInfo.supportUrl;
432     delete[] platformInfo.systemTime;
433 }
434
```

```
435 void DuplicateString(char ** targetString, std::string
    sourceString){
436     *targetString = new char[sourceString.length() + 1];
437     strncpy(*targetString, sourceString.c_str(), (sourceString.
        length() + 1));
438 }
439
440 OCStackResult SetPlatformInfo(std::string platformID, std::string
    manufacturerName,
441         std::string manufacturerUrl, std::string modelNumber, std
            ::string dateOfManufacture,
442         std::string platformVersion, std::string
            operatingSystemVersion,
443         std::string hardwareVersion, std::string firmwareVersion,
             std::string supportUrl,
444         std::string systemTime){
445     DuplicateString(&platformInfo.platformID, platformID);
446     DuplicateString(&platformInfo.manufacturerName,
        manufacturerName);
447     DuplicateString(&platformInfo.manufacturerUrl,
        manufacturerUrl);
448     DuplicateString(&platformInfo.modelNumber, modelNumber);
449     DuplicateString(&platformInfo.dateOfManufacture,
        dateOfManufacture);
450     DuplicateString(&platformInfo.platformVersion,
        platformVersion);
451     DuplicateString(&platformInfo.operatingSystemVersion,
        operatingSystemVersion);
452     DuplicateString(&platformInfo.hardwareVersion,
        hardwareVersion);
453     DuplicateString(&platformInfo.firmwareVersion,
        firmwareVersion);
454     DuplicateString(&platformInfo.supportUrl, supportUrl);
455     DuplicateString(&platformInfo.systemTime, systemTime);
456
457     return OC_STACK_OK;
458 }
459
460 OCStackResult SetDeviceInfo(){
461     OCStackResult result = OC_STACK_ERROR;
462
463     OCResourceHandle handle = OCPlatform::getResourceHandleAtUri(
```

```
        OC_RSRVD_DEVICE_URI);
464     if (handle == NULL){
465         cout << "Falha ao encontrar o recurso! " <<
                OC_RSRVD_DEVICE_URI << endl;
466         return result;
467     }
468
469     result = OCPlatform::bindTypeToResource(handle, deviceType);
470     if (result != OC_STACK_OK){
471         cout << "Falha ao adicionar tipo de dispositivo" << endl;
472         return result;
473     }
474
475     result = OCPlatform::setPropertyValue(PAYLOAD_TYPE_DEVICE,
            OC_RSRVD_DEVICE_NAME, deviceName);
476     if (result != OC_STACK_OK){
477         cout << "Falha ao adicionar o nome do dispositivo" <<
                endl;
478         return result;
479     }
480
481     result = OCPlatform::setPropertyValue(PAYLOAD_TYPE_DEVICE,
            OC_RSRVD_DATA_MODEL_VERSION,
482                                           dataModelVersions);
483     if (result != OC_STACK_OK){
484         cout << "Falha ao adicionar Failed versoes de dados do
                modelo" << endl;
485         return result;
486     }
487
488     result = OCPlatform::setPropertyValue(PAYLOAD_TYPE_DEVICE,
            OC_RSRVD_SPEC_VERSION, specVersion);
489     if (result != OC_STACK_OK){
490         cout << "Falha ao adicionar a versao do spec" << endl;
491         return result;
492     }
493
494     result = OCPlatform::setPropertyValue(PAYLOAD_TYPE_DEVICE,
            OC_RSRVD_PROTOCOL_INDEPENDENT_ID,
495                                           protocolIndependentID);
496     if (result != OC_STACK_OK){
497         cout << "Falha ao adicionar o piid" << endl;
```

```
498         return result;
499     }
500 
501     return OC_STACK_OK;
502 }
503 
504 void * handleSlowResponse (void *param, std::shared_ptr<
    OCResourceRequest> pRequest){
505     // Esta funcao lida com casos de respostas lentas
506     RecursoTemperatura* temperaturaPtr = (RecursoTemperatura*)
        param;
507     // Induzir um caso de resposta lenta usando a funcao sleep
508     std::cout << "Resposta Lenta" << std::endl;
509     sleep (10);
510 
511     auto pResponse = std::make_shared<OC::OCResourceResponse>();
512     pResponse->setRequestHandle(pRequest->getRequestHandle());
513     pResponse->setResourceHandle(pRequest->getResourceHandle());
514     pResponse->setResourceRepresentation(temperaturaPtr->get());
515 
516     pResponse->setResponseResult(OC_EH_OK);
517 
518     // Definir a flag de resposta lenta de volta para falso
519     isSlowResponse = false;
520     OCPlatform::sendResponse(pResponse);
521     return NULL;
522 }
523 
524 void PrintUsage(){
525     std::cout << std::endl;
526     std::cout << "Controle de temperatura do vulcao : Servidor\n"
        ;
527     std::cout << "    Caso a temperatura ultrapasse 120 graus, os
         observadores\n";
528     std::cout << "    Serao notificados sobre um possivel alerta
        \n";
529     std::cout << "    De erupcao vulcanica!\n\n";
530     std::cout << "    A simulacao se iniciara assim que um
        observador se inscrever no recurso.\n\n";
531 }
532 
533 static FILE* client_open(const char* path, const char* mode)
```

```
534 {
535     char const * filename = path;
536     if (0 == strcmp(path, OC_SECURITY_DB_DAT_FILE_NAME))
537     {
538         filename = SVR_DB_FILE_NAME;
539     }
540     else if (0 == strcmp(path, OC_INTROSPECTION_FILE_NAME))
541     {
542         filename = "simpleserver_introspection.dat";
543     }
544     return fopen(filename, mode);
545 }
546
547 int main(int argc, char* argv[]){
548     PrintUsage();
549     OCPersistentStorage ps {client_open, fread, fwrite, fclose,
        unlink };
550
551     if (argc == 1){
552         isListOfObservers = false;
553         isSecure = false;
554     }
555     else if (argc == 2){
556         int value = atoi(argv[1]);
557         switch (value){
558             case 1:
559                 isListOfObservers = true;
560                 isSecure = false;
561                 break;
562             case 2:
563                 isListOfObservers = false;
564                 isSecure = true;
565                 break;
566             case 3:
567                 isListOfObservers = true;
568                 isSecure = true;
569                 break;
570             case 4:
571                 isSlowResponse = true;
572                 break;
573             default:
574                 break;
```

```
575         }
576     }
577     else{
578         return -1;
579     }
580 
581     // Cria um objeto PlatformConfig
582     PlatformConfig cfg {
583         OC::ServiceType::InProc,
584         OC::ModeType::Server,
585         &ps
586     };
587 
588     cfg.transportType = static_cast<OCTransportAdapter>(
        OCTransportAdapter::OC_ADAPTER_IP |
589                                                         OCTransportAdapter
                                                            ::
                                                            OC_ADAPTER_TCP
                                                            );
590     cfg.QoS = OC::QualityOfService::LowQos;
591 
592     OCPlatform::Configure(cfg);
593     OC_VERIFY(OCPlatform::start() == OC_STACK_OK);
594     std::cout << "Iniciando o sevidor e configurando as
        configuracoes de plataforma.\n";
595 
596     OCStackResult result = SetPlatformInfo(gPlatformId,
        gManufacturerName, gManufacturerLink,
597             gModelNumber, gDateOfManufacture, gPlatformVersion,
                gOperatingSystemVersion,
598             gHardwareVersion, gFirmwareVersion, gSupportLink,
                gSystemTime);
599 
600     result = OCPlatform::registerPlatformInfo(platformInfo);
601 
602     if (result != OC_STACK_OK){
603         std::cout << "Registro de plataforma falhou!\n";
604         return -1;
605     }
606 
607     result = SetDeviceInfo();
608 
```

```
609     if (result != OC_STACK_OK){
610         std::cout << "Registro de dispositivo falhou\n";
611         return -1;
612     }
613 
614     try{
615         //Cria a instancia da classe do recurso
616         RecursoTemperatura meuSensor;
617 
618         // Invoke createResource function of class light.
619         meuSensor.criarRecurso();
620         std::cout << "Recurso criado." << std::endl;
621 
622         meuSensor.addType(std::string("core.brighttemperatura"));
623         meuSensor.addInterface(std::string(LINK_INTERFACE));
624         std::cout << "Interface e tipo adicionados." << std::endl
                ;
625 
626         DeletePlatformInfo();
627 
628         std::mutex blocker;
629         std::condition_variable cv;
630         std::unique_lock<std::mutex> lock(blocker);
631         std::cout <<"Aguardando Observadores" << std::endl;
632         cv.wait(lock, []{return false;});
633     }
634     catch(OCException &e){
635         std::cout << "OCException in main : " << e.what() << endl
                ;
636     }
637 
638     OC_VERIFY(OCPlatform::stop() == OC_STACK_OK);
639 
640     return 0;
641 }
```

A.2 Código Cliente

```
1 #include "iotivity_config.h"
2 #ifdef HAVE_UNISTD_H
3 #include <unistd.h>
4 #endif
```

```
5  #ifdef HAVE_PTHREAD_H
6  #include <pthread.h>
7  #endif
8  #ifdef HAVE_WINDOWS_H
9  #include <Windows.h>
10 #endif
11 #include <string>
12 #include <map>
13 #include <cstdlib>
14 #include <mutex>
15 #include <condition_variable>
16 #include "OCPlatform.h"
17 #include "OCApi.h"
18 #include "stdlib.h"
19
20 using namespace OC;
21
22 #define CA_OPTION_CONTENT_VERSION 2053
23 #define COAP_OPTION_CONTENT_FORMAT 12
24 static const char* SVR_DB_FILE_NAME = "./oic_svr_db_client.dat";
25 typedef std::map<OCResourceIdentifier, std::shared_ptr<OCResource
      >> DiscoveredResourceMap;
26
27 DiscoveredResourceMap discoveredResources;
28 std::shared_ptr<OCResource> curResource;
29 static ObserveType OBSERVE_TYPE_TO_USE = ObserveType::Observe;
30 static OCConnectivityType TRANSPORT_TYPE_TO_USE =
      OCConnectivityType::CT_ADAPTER_IP;
31 std::mutex curResourceLock;
32
33 class RecursoTemperatura{
34 public:
35
36     bool m_estado;
37     int m_temperatura;
38     std::string m_nome;
39
40     RecursoTemperatura() : m_estado(false), m_temperatura(0),
          m_nome(""){
41     }
42 };
43
```

```
44 RecursoTemperatura sensor;
45
46 int observe_count(){
47     static int oc = 0;
48     return ++oc;
49 }
50
51 void onObserve(const HeaderOptions /*Opcoes do cabecalho*/, const
       OCRepresentation& rep,
52                     const int& eCode, const int& sequenceNumber){
53     try{
54         if(eCode == OC_STACK_OK && sequenceNumber <=
              MAX_SEQUENCE_NUMBER){
55             if(sequenceNumber == OC_OBSERVE_REGISTER){
56                 std::cout << "Registro de observacao bem sucedida
                      !" << std::endl;
57             }
58
59             std::cout << "Resultado da Observacao:"<<std::endl;
60             rep.getValue("Estado", sensor.m_estado);
61             rep.getValue("Temperatura", sensor.m_temperatura);
62             rep.getValue("Nome", sensor.m_nome);
63
64             system("cls");
65             std::cout << "\tNome: " << sensor.m_nome << std::endl
                  ;
66             std::cout << "\tEstado: " << sensor.m_estado << std::
                  endl;
67             if (sensor.m_temperatura > 120){
68                 std::cout << "\n\t***Alerta*** A temperatura esta
                      aumentando! Perigo de erupcao vulcanica!!!" <<
                      std::endl;
69             }
70             std::cout << "\tTemperatura: " << sensor.
                  m_temperatura << std::endl;
71
72             if(observe_count() == 11){
73                 std::cout<<"Cancelando Observacao..."<<std::endl;
74                 OCStackResult result = curResource->cancelObserve
                      ();
75
76                 std::cout << "Resultado: "<< result <<std::endl;
```

```
77                  sleep(5);
78                  std::cout << "Cancelado com sucesso!"<<std::endl;
79                  std::exit(0);
80              }
81          }
82          else{
83              if(eCode == OC_STACK_OK){
84                  std::cout << "Cancelando..." << std::endl;
85              }
86              else{
87                  std::cout << "Reposta de erro onObserve: " <<
                        eCode << std::endl;
88                  std::exit(-1);
89              }
90          }
91      }
92      catch(std::exception& e){
93          std::cout << "Exception: " << e.what() << " Em onObserve"
                << std::endl;
94      }
95
96  }
97
98  void onPost2(const HeaderOptions& /*headerOptions*/,
99          const OCRepresentation& rep, const int eCode){
100     try{
101         if(eCode == OC_STACK_OK || eCode ==
                OC_STACK_RESOURCE_CREATED
102                 || eCode == OC_STACK_RESOURCE_CHANGED){
103             std::cout << "Requisicao POST realizada com sucesso"
                    << std::endl;
104
105             if(rep.hasAttribute("createduri")){
106                 std::cout << "\tUri do recurso criado: "
107                     << rep.getValue<std::string>("createduri") <<
                            std::endl;
108             }
109             else{
110                 rep.getValue("Estado", sensor.m_estado);
111                 rep.getValue("Temperatura", sensor.m_temperatura)
                        ;
112                 rep.getValue("Nome", sensor.m_nome);
```

```
113
114                 std::cout << "\tEstado: " << sensor.m_estado <<
                        std::endl;
115                 std::cout << "\tTemperatura: " << sensor.
                        m_temperatura << std::endl;
116                 std::cout << "\tNome: " << sensor.m_nome << std::
                        endl;
117             }
118
119             if (OBSERVE_TYPE_TO_USE == ObserveType::Observe)
120                 std::cout << std::endl << "Tipo de observacao:
                        ObserveType." << std::endl << std::endl;
121             else if (OBSERVE_TYPE_TO_USE == ObserveType::
                    ObserveAll)
122                 std::cout << std::endl << "Tipo de observacao:
                        ObserveAll." << std::endl << std::endl;
123
124             curResource->observe(OBSERVE_TYPE_TO_USE,
                    QueryParamsMap(), &onObserve);
125
126         }
127         else{
128             std::cout << "onPost2 Response error: " << eCode <<
                    std::endl;
129             std::exit(-1);
130         }
131     }
132     catch(std::exception& e){
133         std::cout << "Exception: " << e.what() << " Em onPost2"
                << std::endl;
134     }
135
136 }
137
138 void onPost(const HeaderOptions& /*Opcoes de cabecalho*/,
139         const OCRepresentation& rep, const int eCode){
140     try{
141         if(eCode == OC_STACK_OK || eCode ==
                OC_STACK_RESOURCE_CREATED
142                 || eCode == OC_STACK_RESOURCE_CHANGED){
143             std::cout << "Requisicao POST realizada com sucesso"
                    << std::endl;
```

```
144
145             if(rep.hasAttribute("createduri")){
146                 std::cout << "\tUri do recurso criado: "
147                     << rep.getValue<std::string>("createduri") <<
                        std::endl;
148             }
149             else{
150                 rep.getValue("Estado", sensor.m_estado);
151                 rep.getValue("Temperatura", sensor.m_temperatura)
                    ;
152                 rep.getValue("Nome", sensor.m_nome);
153
154                 std::cout << "\tEstado: " << sensor.m_estado <<
                    std::endl;
155                 std::cout << "\tTemperatura: " << sensor.
                    m_temperatura << std::endl;
156                 std::cout << "\tNome: " << sensor.m_nome << std::
                    endl;
157             }
158
159             OCRepresentation rep2;
160
161             std::cout << "Colocando a representacao do sensor..."
                <<std::endl;
162
163             sensor.m_estado = true;
164             sensor.m_temperatura = 55;
165
166             rep2.setValue("Estado", sensor.m_estado);
167             rep2.setValue("power", sensor.m_temperatura);
168
169             curResource->post(rep2, QueryParamsMap(), &onPost2);
170         }
171         else{
172             std::cout << "onPost resposta de erro: " << eCode <<
                std::endl;
173             std::exit(-1);
174         }
175     }
176     catch(std::exception& e){
177         std::cout << "Exception: " << e.what() << " Em onPost" <<
                std::endl;
```

```
178     }
179 }
180
181 // Funcao local para colocar um estado diferente para o recurso
    sensor
182 void colocandoNovaTemperatura(std::shared_ptr<OCResource>
    resource){
183     if(resource){
184         OCRepresentation rep;
185
186         std::cout << "Colocando a representacao do sensor..."<<
              std::endl;
187
188         sensor.m_estado = false;
189         sensor.m_temperatura = 105;
190
191         rep.setValue("Estado", sensor.m_estado);
192         rep.setValue("Temperatura", sensor.m_temperatura);
193
194         resource->post(rep, QueryParamsMap(), &onPost);
195     }
196 }
197
198 // callback handler de requisicoes put
199 void onPut(const HeaderOptions& /*Opcoes de cabecalho*/, const
    OCRepresentation& rep, const int eCode){
200     try{
201         if (eCode == OC_STACK_OK || eCode ==
              OC_STACK_RESOURCE_CHANGED){
202             std::cout << "Requisicao PUT realizada com sucesso"
                  << std::endl;
203
204             rep.getValue("Estado", sensor.m_estado);
205             rep.getValue("Temperatura", sensor.m_temperatura);
206             rep.getValue("Nome", sensor.m_nome);
207
208             std::cout << "\tEstado: " << sensor.m_estado << std::
                  endl;
209             std::cout << "\tTemperatura: " << sensor.
                  m_temperatura << std::endl;
210             std::cout << "\tNome: " << sensor.m_nome << std::endl
                  ;
```

```
211
212             colocandoNovaTemperatura(curResource);
213         }
214         else{
215             std::cout << "onPut resposta de erro: " << eCode <<
                    std::endl;
216             std::exit(-1);
217         }
218     }
219     catch(std::exception& e){
220         std::cout << "Exception: " << e.what() << " Em onPut" <<
                std::endl;
221     }
222 }
223
224 // Funcao local para colocar um estado diferente para o recurso
    sensor
225 void putTemperatura(std::shared_ptr<OCResource> resource){
226     if(resource){
227         OCRepresentation rep;
228
229         std::cout << "Colocando a representacao do sensor..."<<
                std::endl;
230
231         sensor.m_estado = true;
232         sensor.m_temperatura = 15;
233
234         rep.setValue("Estado", sensor.m_estado);
235         rep.setValue("Temperatura", sensor.m_temperatura);
236
237         resource->put(rep, QueryParamsMap(), &onPut);
238     }
239 }
240
241 // Callback handler em requisicoes GET
242 void onGet(const HeaderOptions& headerOptions, const
    OCRepresentation& rep, const int eCode){
243     try{
244         if(eCode == OC_STACK_OK){
245             std::cout << "Requisicao GET feita com sucesso!" <<
                    std::endl;
246             std::cout << "URI do Recurso: " << rep.getUri() <<
```

```
                std::endl;
247
248             // Pegando as opcoes de cabecalho
249             if ( headerOptions.size() == 0){
250                 std::cout << "Sem opcoes de cabecalho" << std::
                        endl;
251             }
252             else{
253                 for (auto it = headerOptions.begin(); it !=
                        headerOptions.end(); ++it){
254                     if (it->getOptionID() ==
                            COAP_OPTION_CONTENT_FORMAT){
255                         size_t dataLength = it->getOptionData().
                                length();
256                         char* optionData = new char[dataLength];
257                         strncpy(optionData, it->getOptionData().
                                c_str(), dataLength);
258                         int format = optionData[0] * 256 +
                                optionData[1];
259                         std::cout << "Formato do servidor na
                                resposta GET:" << format << std::endl;
260                         delete[] optionData;
261                     }
262                     if (it->getOptionID() ==
                            CA_OPTION_CONTENT_VERSION){
263                         size_t dataLength = it->getOptionData().
                                length();
264                         char* optionData = new char[dataLength];
265                         strncpy(optionData, it->getOptionData().
                                c_str(), dataLength);
266                         int version = optionData[0] * 256;
267                         std::cout << "Versao do servidor na
                                reposta GET:" << version << std::endl;
268                         delete[] optionData;
269                     }
270                 }
271             }
272             rep.getValue("Estado", sensor.m_estado);
273             rep.getValue("Temperatura", sensor.m_temperatura);
274             rep.getValue("Nome", sensor.m_nome);
275
276             std::cout << "\tEstado: " << sensor.m_estado << std::
```

```
                endl;
277             std::cout << "\tTemperatura: " << sensor.
                    m_temperatura << std::endl;
278             std::cout << "\tNome: " << sensor.m_nome << std::endl
                    ;
279
280             putTemperatura(curResource);
281         }
282         else{
283             std::cout << "onGET resposta de erro: " << eCode <<
                    std::endl;
284             std::exit(-1);
285         }
286     }
287     catch(std::exception& e){
288         std::cout << "Exception: " << e.what() << " Em onGet" <<
                std::endl;
289     }
290 }
291
292 // Funcao local para pegar a representacao do sensor
293 void pegaRepresentacaoSensor(std::shared_ptr<OCResource> resource
    ){
294     if(resource){
295         std::cout << "Pegando a representacao do sensor..."<<std
                ::endl;
296
297         QueryParamsMap test;
298         resource->get(test, &onGet);
299     }
300 }
301
302 // Callback para procurar recursos
303 void foundResource(std::shared_ptr<OCResource> resource){
304     std::cout << "Procurando Recursos:\n";
305     std::string resourceURI;
306     std::string hostAddress;
307     try{
308         {
309             std::lock_guard<std::mutex> lock(curResourceLock);
310             if(discoveredResources.find(resource->
                    uniqueIdentifier()) == discoveredResources.end()){
```

```
311                 std::cout << "Recurso encontrado " << resource->
                        uniqueIdentifier() <<
312                      " Pela primeira vez no servidor com ID: "<<
                            resource->sid()<<std::endl;
313                 discoveredResources[resource->uniqueIdentifier()]
                         = resource;
314             }
315             else{
316                 std::cout<<"Recurso encontrado "<< resource->
                        uniqueIdentifier() << " novamente!"<<std::endl;
317             }
318
319             if(curResource){
320                 std::cout << "Procurando outro recurso, ignorando
                        "<<std::endl;
321                 return;
322             }
323         }
324
325         // Faz algumas operacoes com objeto do recurso.
326         if(resource){
327             std::cout<<"Recurso Descoberto:"<<std::endl;
328             // Pega o URI do recurso
329             resourceURI = resource->uri();
330             std::cout << "\tURI do recurso: " << resourceURI <<
                    std::endl;
331
332             // Pega o endereco do hosto pertencente ao recurso
333             hostAddress = resource->host();
334             std::cout << "\tEndereco do host pertencente ao
                    recurso: " << hostAddress << std::endl;
335
336             // Pega os tipos do recurso
337             std::cout << "\tLista dos tipos do recurso: " << std
                    ::endl;
338             for(auto &resourceTypes : resource->getResourceTypes
                    ()){
339                 std::cout << "\t\t" << resourceTypes << std::endl
                        ;
340             }
341
342             // Pega as interfaces do recurso
```

```
343             std::cout << "\tLista das interfaces do recurso: " <<
                    std::endl;
344             for(auto &resourceInterfaces : resource->
                    getResourceInterfaces()){
345                 std::cout << "\t\t" << resourceInterfaces << std
                        ::endl;
346             }
347
348             // Pega o host do recurso
349             std::cout << "\tHost do recurso: " << std::endl;
350             std::cout << "\t\t" << resource->host() << std::endl;
351
352             // Pega as informacoes dos terminais do recurso
353             std::cout << "\tLista dos terminais do recurso: " <<
                    std::endl;
354             for(auto &resourceEndpoints : resource->getAllHosts()
                    ){
355                 std::cout << "\t\t" << resourceEndpoints << std::
                        endl;
356             }
357
358             // Se o recurso for encontrado no adaptador baseado
                    em ip.
359             if (std::string::npos != resource->host().find("coap
                    ://") ||
360                 std::string::npos != resource->host().find("coaps
                        ://") ||
361                 std::string::npos != resource->host().find("coap+
                        tcp://") ||
362                 std::string::npos != resource->host().find("coaps
                        +tcp://")){
363                 for(auto &resourceEndpoints : resource->
                        getAllHosts()){
364                     if (resourceEndpoints.compare(resource->host
                            ()) != 0 &&
365                         std::string::npos == resourceEndpoints.
                            find("coap+rfcomm")){
366                         std::string newHost = resourceEndpoints;
367
368                         if (std::string::npos != newHost.find("
                            tcp")){
369                             TRANSPORT_TYPE_TO_USE =
```

```
                                OCConnectivityType::CT_ADAPTER_TCP;
370                         }
371                         else{
372                             TRANSPORT_TYPE_TO_USE =
                                    OCConnectivityType::CT_ADAPTER_IP;
373                         }
374                         // Muda o host do recurso se existir
                            outro host
375                         std::cout << "\tAlterando o host dos
                            terminais do recurso" << std::endl;
376                         std::cout << "\t\t" << "Host atual e "
377                                 << resource->setHost(newHost)
                                        << std::endl;
378                         break;
379                     }
380                 }
381             }
382
383             if(resourceURI == "/a/temperatura"){
384                 HeaderOptions headerOptions = resource->
                        getServerHeaderOptions();
385                 if (headerOptions.size() == 0){
386                     std::cout << "Sem opcoes de cabecalho" << std
                            ::endl;
387                 }
388                 else{
389                     for (auto it = headerOptions.begin(); it !=
                            headerOptions.end(); ++it){
390                         if (it->getOptionID() ==
                                COAP_OPTION_CONTENT_FORMAT){
391                             size_t dataLength = it->getOptionData
                                    ().length();
392                             char* optionData = new char[
                                    dataLength];
393                             strncpy(optionData, it->getOptionData
                                    ().c_str(), dataLength);
394                             int format = optionData[0] * 256 +
                                    optionData[1];
395                             std::cout << "Formato do servidor na
                                    resposta de descoberta:" << format
396                                         << std::endl;
397                             delete[] optionData;
```

```
398                         }
399                         if (it->getOptionID() ==
                               CA_OPTION_CONTENT_VERSION){
400                             size_t dataLength = it->getOptionData
                                   ().length();
401                             char* optionData = new char[
                                   dataLength];
402                             strncpy(optionData, it->getOptionData
                                   ().c_str(), dataLength);
403                             int version = optionData[0] * 256;
404                             std::cout << "Versao do servidor na
                                   reposta de descoberta:" << version
405                                     << std::endl;
406                             delete[] optionData;
407                         }
408                     }
409                 }
410
411                 if (resource->connectivityType() &
                       TRANSPORT_TYPE_TO_USE){
412                     curResource = resource;
413                     // Pega o endereco do hosto pertencente ao
                           recurso
414                     std::cout << "\tAddress of selected resource:
                           " << resource->host() << std::endl;
415
416                     // Chama uma funcao local que invocara
                           internamente a API get no ponteiro do
                           recurso
417                     pegaRepresentacaoSensor(resource);
418                 }
419             }
420         }
421         else{
422             // Recurso invalido
423             std::cout << "Este recurso e invalido" << std::endl;
424         }
425
426     }
427     catch(std::exception& e){
428         std::cerr << "Exception em foundResource: "<< e.what() <<
              std::endl;
```

```
429     }
430 }
431
432 void printUsage(){
433     std::cout << std::endl;
434     std::cout << "
           ----------------------------------------------------------------
           n";
435     std::cout << "Use : simpleclient para comecar a simulacao" <<
            std::endl;
436     std::cout << "
           ----------------------------------------------------------------
           n\n";
437 }
438
439 void checkObserverValue(int value){
440     if (value == 1){
441         OBSERVE_TYPE_TO_USE = ObserveType::Observe;
442         std::cout << "<===Colocando o tipo de observacao em
               Observe===>\n\n";
443     }
444     else if (value == 2){
445         OBSERVE_TYPE_TO_USE = ObserveType::ObserveAll;
446         std::cout << "<===Colocando o tipo de observacao em
               ObserveAll===>\n\n";
447     }
448     else{
449         std::cout << "<===Tipo de observacao invalido!"
450                   <<" Configurando o tipo de observacao em
                        Observe===>\n\n";
451     }
452 }
453
454 void checkTransportValue(int value){
455     if (1 == value){
456         TRANSPORT_TYPE_TO_USE = OCConnectivityType::CT_ADAPTER_IP
               ;
457         std::cout << "<===Configurando o tipo de transporte para
               IP===>\n\n";
458     }
459     else if (2 == value){
460         TRANSPORT_TYPE_TO_USE = OCConnectivityType::
```

```
            CT_ADAPTER_TCP;
461         std::cout << "<===Configurando o tipo de transporte para
               TCP===>\n\n";
462     }
463     else{
464         std::cout << "<===Tipo de transporte selecionado e
               invalido."
465                 <<" Configurando o tipo de transporte para IP
                       ===>\n\n";
466     }
467 }
468
469 static FILE* client_open(const char* path, const char* mode){
470     if (0 == strcmp(path, OC_SECURITY_DB_DAT_FILE_NAME)){
471         return fopen(SVR_DB_FILE_NAME, mode);
472     }
473     else{
474         return fopen(path, mode);
475     }
476 }
477
478 int main(int argc, char* argv[]) {
479
480     std::ostringstream requestURI;
481     OCPersistentStorage ps {client_open, fread, fwrite, fclose,
           unlink };
482     try{
483         printUsage();
484         if (argc == 1){
485             std::cout << "<===Configurando o tipo de observacao
                   como Observe e Tipo de conexao como IP===>\n\n";
486         }
487         else if (argc == 2){
488             checkObserverValue(std::stoi(argv[1]));
489         }
490         else if (argc == 3){
491             checkObserverValue(std::stoi(argv[1]));
492             checkTransportValue(std::stoi(argv[2]));
493         }
494         else{
495             std::cout << "<===Numero invalido de argumentos de
                   linha de comando===>\n\n";
```

```
496             return -1;
497         }
498     }
499     catch(std::exception& )
500     {
501         std::cout << "<===Argumentos invalidos===>\n\n";
502         return -1;
503     }
504
505     // Create PlatformConfig object
506     PlatformConfig cfg {
507         OC::ServiceType::InProc,
508         OC::ModeType::Both,
509         &ps
510     };
511
512     cfg.transportType = static_cast<OCTransportAdapter>(
        OCTransportAdapter::OC_ADAPTER_IP |
513                                                         OCTransportAdapter
                                                            ::
                                                            OC_ADAPTER_TCP
                                                            );
514     cfg.QoS = OC::QualityOfService::HighQos;
515
516     OCPlatform::Configure(cfg);
517     try{
518         OC_VERIFY(OCPlatform::start() == OC_STACK_OK);
519
520         // faz com que todos os valores booleanos sejam impressos
                como 'verdadeiro / falso' neste fluxo
521         std::cout.setf(std::ios::boolalpha);
522         // Encontra todos os recursos
523         requestURI << OC_RSRVD_WELL_KNOWN_URI;// << "?rt=core.
            temperatura";
524
525         OCPlatform::findResource("", requestURI.str(),
526                 CT_DEFAULT, &foundResource);
527         std::cout<< "Procurando Recursos... " <<std::endl;
528
529         // Encontrar recurso e feito duas vezes para descobrirmos
                os recursos originais uma segunda vez.
530         // Esses recursos terao o mesmo uniqueidentifier (ainda
```

```
            que sejam objetos diferentes), de modo que
531         // podemos verificar / mostrar o codigo de verificacao
            duplicado em foundResource (acima);
532         OCPlatform::findResource("", requestURI.str(),
533                 CT_DEFAULT, &foundResource);
534         std::cout<< "Procurando recursos pela segunda vez..." <<
            std::endl;
535
536         std::mutex blocker;
537         std::condition_variable cv;
538         std::unique_lock<std::mutex> lock(blocker);
539         cv.wait(lock);
540
541         OC_VERIFY(OCPlatform::stop() == OC_STACK_OK);
542
543     }catch(OCException& e)
544     {
545         oclog() << "Exception in main: "<<e.what();
546     }
547
548     return 0;
549 }
```