Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica e Informática

Projeto e Desenvolvimento de um Oxímetro Conectado e Interoperável Utilizando o Protocolo de Comunicação OCF

Henrique Jordão Figueiredo Alves

Campina Grande, PB
Dezembro de 2018

Henrique Jordão Figueiredo Alves

Projeto e Desenvolvimento de um Oxímetro Conectado e Interoperável Utilizando o Protocolo de Comunicação OCF

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Área de Concentração: Controle e Automação

Orientador: Danilo Freire de Souza Santos

Campina Grande, PB
Dezembro de 2018

Henrique Jordão Figueiredo Alves

Projeto e Desenvolvimento de um Oxímetro Conectado e Interoperável Utilizando o Protocolo de Comunicação OCF

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovado em ___/___ /___

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Danilo Freire de Souza Santos
Universidade Federal de Campina Grande
Orientador

Agradecimentos

Gostaria de agradecer primeiramente a Deus, que apesar de todas as dificuldades sempre esteve ao meu lado e garantiu que tudo pudesse dar certo no final.

Aos meus pais, Alexandro Alves e Maria Solange, por sempre terem feito de tudo para me fornecer a melhor formação possível e sempre demonstrarem apoio quanto às minhas decisões.

Ao meu irmão Hugo, pelas alegrias que este me proporciona desde o seu nascimento.

Aos meus colegas de curso, pelo companheirismo e pela ajuda mútua na qual compartilhamos ao longo do curso. Cito alguns, Jefferson Albuquerque, Thiago Ribeiro Félix, Marcelo Meneses, Michael Douglas, Leonardo Dantas, Matheus Andrade, Jesney Pires, dentre outros.

Ao meu orientador, profº Danilo Freire, pela ajuda e paciência ao logno do desenvolvimento do projeto e por ter transmitido o máximo de conhecimento possível que sem dúvida irá contribuir no meu crescimento profissional.

”Eu não tenho tempo para uma esposa e um avião.”
Wilbur Wright

Resumo

Com o avanço tecnológico que ocorre no dia-a-dia das pessoas, torna-se mais iminente a importância do estudo da Internet das Coisas e do incentivo ao desenvolvimento de projetos de modelos interoperáveis utilizando este conceito. A área da saúde é um dos setores onde a tendência da utilização de novas tecnologias para melhorar o bem-estar da população, vem se tornando um desafio para os profissionais envolvidos nessa questão. Com a presença do arcabouço OCF, que compõe-se de uma gama de protocolos de rede e comunicação, esta tese pode vir a sair do papel e passar a integrar os ambientes que fazem parte do campo da engenharia biomédica. Neste trabalho foi realizado um projeto que utiliza a tecnologia OCF para criar um dispositivo de saúde interoperável para Internet das Coisas.

Palavras chave: Internet das Coisas, interoperável, OCF, engenharia biomédica, protocolos de rede.

Abstract

With the technological advancement that occurs in people's daily lives, it becomes more imminent the importance of studying the Internet of Things and encouraging the development of interoperable model projects using this concept. The health area is one of the sectors where the tendency of the use of new technologies to improve the well-being of the population, has become a challenge for the professionals involved in this issue. With the presence of the OCF framework, which consists of a range of network and communication protocols, this thesis may come out of the paper and integrate into the environments that are part of the field of biomedical engineering. In this work it was carried out a project that uses OCF technology to create an interoperable health device for Internet of Things.

Keywords: Internet of Things, interoperable, OCF, biomedical engineering, network protocols.

Lista de Figuras

Lista de Abreviaturas e Siglas

bpm Batimentos por Minuto

CoAP Constrained Application Protocol

CRUDN CREATE, RETRIEVE, UPDATE, DELETE, NOTIFY

HTTP Hypertext Transfer Protocol

IoT Internet das Coisas

IP Internet Protocol

IPC Inter-Process Communication

JSON JavaScript Object Notation

LAN Local Area Network

M2M Machine-to-Machine

OCF Open Connectivity Foundation

REST Representational State Transfer

RFC Request for Comments

ROA Resource-oriented Architecture

SoC System on a Chip

SpO_2 Nível de Oxigênio no Sangue

SSH Secure Shell

TCP Transmission Control Protocol

UDP User Datagram Protocol

URI Uniform Resource Identifier

Sumário

1 Introdução

Há pouco mais de 50 anos atrás eram elaborados os primeiros projetos de comutação de pacotes entre computadores por meio de pesquisas e estudos feitos em algumas universidades americanas, dando origem à rede de comunicação mundial que conhecemos hoje como **Internet**. Desde os primeiros protocolos de intercomunicação de redes implementados junto à ARPANET (*Advanced Research Projects Agency Network*), até os modelos atuais, é notável perceber a importância pela qual esta rede ampla exerce na vida e no dia-a-dia das pessoas.

Existem diversas redes espalhadas pelo mundo, e para que as pessoas realizem seu desejo de se comunicarem, é necessário que se estabeleçam conexões entre essas redes [1]. Hoje em dia, com a evolução tecnológica dos celulares, boa parte da população possui um dispositivo móvel e se encontra conectada 24 horas por dia na rede global, através de redes móveis sem fio como Wi-Fi ou redes celulares (ex: 3G, 4G, etc).

Muito em breve, esse cenário irá se expandir para a conexão de outros objetos além dos celulares, tablets, televisores ou computadores com a rede. Com o surgimento de tecnologias que oferecem serviços de computação em nuvem (Ex: Amazon AWS)[1] e técnicas de roteamento e detecção de sinal *wireless*, o mundo se encaminha para a chamada 4ª Revolução Industrial, com o surgimento da **Internet das Coisas**, ou IoT (*Internet of Things*).

De acordo com Francis DaCosta, o surgimento da Internet das Coisas destrói toda noção precedente e preconcebida da arquitetura de rede. Até hoje, as redes foram inventadas por engenheiros especializados em protocolos e teoria de roteamento. Mas a arquitetura da Internet das Coisas dependerá muito mais de lições derivadas da natureza do que esquemas de rede tradicionais [2].

É perceptível que com a chegada da IoT, vários estabelecimentos do segundo setor irão ser afetados com essa nova tecnologia, e isso inclui a área da sáude. Este novo fator compõe a chamada engenharia biomédica, que é voltada para a concepção de equipamentos utilizados em tratamentos, terapias e diagnósticos.

Melhorar a eficiência dos serviços e da infraestrutura de cuidados de saúde e sistemas biomédicos é um dos mais desafiadores objetivos da sociedade moderna. De fato, a necessidade de entregar cuidados de qualidade aos pacientes, reduzindo os custos de saúde e, ao mesmo tempo, enfrentar o problema da escassez de enfermeiros pessoais, é uma questão importante [3].

Para isso ser possível, é preciso que haja um sistema de troca de informações e dados estável entre o servidor e cliente, para estabelecer a comunicação entre o usuário, no caso o profissional de saúde, e a rede. Essa troca de informações é realizada através de **protocolos de comunicação interoperáveis**. A interoperabilidade em nível de rede lida com mecanismos para permitir a troca de mensagens entre sistemas através de diferentes redes (redes de redes) para comunicação de

[1] https://aws.amazon.com/pt/

ponta a ponta. Para tornar os sistemas interoperáveis, cada sistema deve ser capaz de compartilhar informações e trocar mensagens com outros sistemas através de vários tipos de redes. Devido ao ambiente de rede dinâmico e heterogêneo em IoT, o nível de interoperabilidade de rede deve tratar de problemas como endereçamento, roteamento, otimização de recursos, segurança, Qualidade de Serviço (do inglês, *Quality of Service*) e suporte à mobilidade [4]. A necessidade de protocolos de comunicação interoperáveis, como o OCF (*Open Connectivity Foundation*), é essencial para a realização desses serviços.

O protocolo OCF, garante aos consumidores desses tipos de serviço uma interoperabilidade segura que permite aos dispositivos se comunicarem independentemente de sua forma, sistema operacional, provedor de serviços, tecnologia de transporte ou ecossistema. Algumas das principais empresas do mundo, como a Cisco, Microsoft, LG, Samsung, dentre outras, adotam o OCF como ferramenta para comunicação interoperável em Internet das Coisas. [5]

Esta tecnologia oferece aos profissionais do ramo da saúde, novas possibilidades de resolução de problemas, assim como a otimização de alguns procedimentos médicos adotados em certas situações de emergência. Essa prática de utilização de informações de saúde é conhecida como *e-Health*.

Baseando-se nos conceitos de Internet das Coisas, utilizando o protocolo OCF e a tecnologia *e-Health* para um Raspberry Pi, é possível realizar um modelo de medição, envio e captura de dados de um oxímetro e utilizar a rede para compartilhar informações com sistemas de monitoramento remoto de pacientes remotos, por exemplo, com sistemas em nuvem.

Desta forma, os profissionais da área de saúde podem conferir os resultados dos exames cardíacos dos pacientes sem precisar estar, necessariamente, no mesmo local dos próprios. Esses dados podem ser acessados através de uma aplicação Web, dispositivos móveis ou computadores pessoais.

1.1 Objetivos

O objetivo deste trabalho de conclusão de curso é a de projetar um protótipo de um dispositivo de saúde pessoal, do tipo oxímetro, conectado a um Raspberry Pi Modelo B, para a Internet das Coisas. Para estabelecer uma conexão cliente-servidor entre o oxímetro (servidor) e o computador (cliente), foi utilizado um framework código aberto chamado Iotivity, que possibilita a comunicação entre ambos. Este framework foi desenvolvido pela Open Connectivity Foundation^{TM}e utiliza o protocolo OCF.

Portanto, os objetivos específicos desse trabalho são:

- Pesquisar e analisar o modelo de comunicação do OCF e como ele se adapta a dispositivos pessoais de saúde;

- Avaliar e implementar o funcionamento de um dispositivo OCF em uma plataforma embarcada de referência;
- Validar o uso do dispositivo OCF desenvolvido com uma aplicação de exemplo.

1.2 Estrutura e Organização do Documento

Este documento foi organizado da seguinte forma:

- **Capítulo 1 - Introdução**: apresentação da motivação e problemática do trabalho.
- **Capítulo 2 - Revisão Bibliográfica e Tecnológica**: as tecnologias utilizadas no trabalho.
- **Capítulo 3 - Proposta e Solução**: apresentação do projeto, seus requisitos e detalhes de como foi definida a solução para o problema.
- **Capítulo 4 - Desenvolvimento**: detalhamento de como o projeto foi implementado.
- **Capítulo 5 - Validação**: detalhamento de como o projeto foi testado e validado.
- **Conclusão**: resumo do que foi realizado, apresentação dos resultados e projeções para o futuro do tema.
- **Referências**: referências bibliográficas utilizadas no documento.

2 Revisão Bibliográfica e Tecnológica

Esse capítulo detalha os conceitos teóricos, bem como as tecnologias utilizadas para a implementação da solução ao problema proposto.

2.1 Princípios de Redes de Computadores

É impossível dissertar sobre a Internet das Coisas ou os seus serviços sem o conhecimento prévio sobre as redes de computadores. Uma rede de computadores é dada por um conjunto de nós de conexão, ou *hosts*, que trocam informações e dados entre si. Exemplos de *hosts* podem incluir computadores, celulares, tablets, servidores, etc.

A comunicação entre redes, muitas vezes se dá por uma relação **cliente-servidor**. Esta relação ocorre quando temos um *host*, devidamente conectado numa rede, atuando no compartilhamento de serviço e recursos para os demais *hosts* da rede, fazendo dele um **servidor**, e um outro *host*, que realiza a solicitação desses recursos compartilhados, fazendo dele um **cliente**. O projeto deste trabalho, baseia-se nessa relação.

A arquitetura de uma rede de computadores pode ser organizada em uma pilha de protocolos dividida em camadas. Cada camada representa uma determinada função na rede, que por sua vez irão conter seus devidos protocolos característicos operando na comunicação.Um dos modelos de comunicação em camadas mais conhecido é o **Modelo TCP/IP**.

Mas afinal, o que é um protocolo? Pela sua definição, um protocolo é dado como um conjunto de normas ou procedimentos que devem ser respeitados. Em uma rede de computadores não é diferente. Um protocolo de rede é semelhante a um protocolo humano, a única diferença é que as entidades que trocam mensagens e realizam ações são componentes de hardware ou software de algum dispositivo. Todas as atividades na Internet que envolvem duas ou mais entidades remotas comunicantes são governadas por um protocolo [6]. Portanto, um protocolo de rede nada mais é do que um conjunto de regras que irão compor a comunicação entre os *hosts* conectados em uma rede.

2.1.1 Modelo TCP/IP

A arquitetura do modelo TCP/IP possui ao todo cinco camadas:

- **Camada de Aplicação**: onde residem aplicações de rede e seus protocolos.
- **Camada de Transporte**: carrega mensagens da camada de aplicação entre os lados do cliente e servidor de uma aplicação.
- **Camada de Rede**: responsável pelo transporte de datagramas.

- **Camada de Enlace**: onde se transmitem os pacotes do nó de origem até o nó de destino da conexão.
- **Camada Física**: transporte bit a bit entre a origem e o destino.

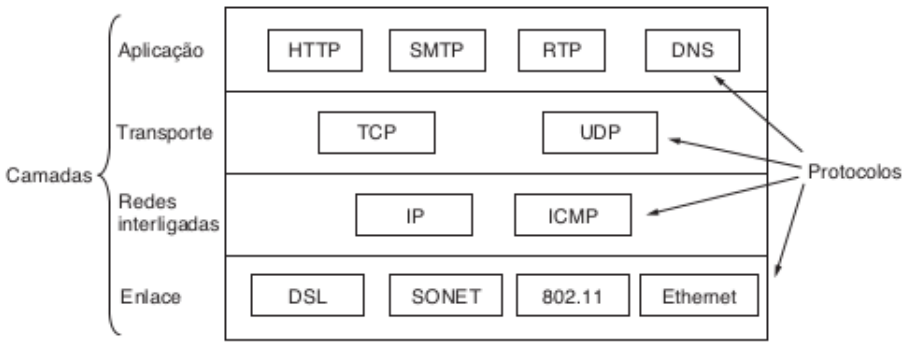

Figura 2.1: Modelo TCP/IP (TANENBAUM, 2011)

2.1.2 Os Protocolos de Rede

Como ilustrado no diagrama da Figura 2.1, existem diversos protocolos com funções diferentes, de acordo com cada camada do modelo TCP/IP. Nesta seção serão apresentados os protocolos que serão mais importantes para o entendimento do trabalho.

- ***Internet Protocol* - IP**

O protocolo IP (*Internet Protocol*) é, sem dúvida, o mais importante do modelo TCP/IP. É responsável pela maior parte do tráfego de dados e informações na Internet. Sem ele, não seria possível estabelecer uma comunicação entre os nós interligados. Ele pertence à camada de rede, e atualmente, possui duas versões: o IPv4 e o IPv6.

Em uma rede TCP/IP, cada computador é identificado por um endereço IP. Levando-se em conta a versão IPv4, esse endereço IP consiste em quatro conjuntos de 8 bits, chamados de octetos [8]. Normalmente, esses octetos são representados em números decimais, possibilitando assim, uma melhor leitura do endereço, como apresentado na Figura 2.2.

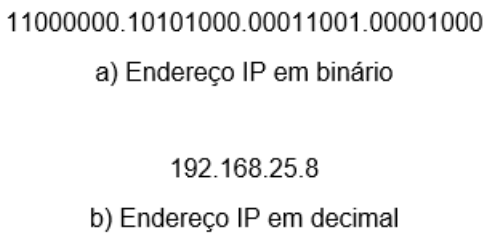

Figura 2.2: Exemplo de endereçamento IP

Os endereços IP funcionam como se fosse a caixa postal dos dispositivos ligados na rede. É a partir dele que se dá o envio e recebimento de pacotes entre os nós.

- ***Transmission Control Protocol* - TCP**

O protocolo TCP (*Transmission Control Protocol*) pertence à camada de transporte da família TCP/IP. É um protocolo para transmissão de mensagens e pacotes oriundos da camada de aplicação e provê serviços orientados a conexão para suas aplicações. Alguns desses serviços são a garantia de entrega de mensagens da camada de aplicação ao destino. Segundo Kurose, o TCP também fragmenta mensagens longas em segmentos mais curtos e provê mecanismo de controle de congestionamento, de modo que uma origem reduz sua velocidade de transmissão quando a rede está congestionada [6].

- ***User Datagram Protocol* - UDP**

O UDP (*User Datagram Protocol*) também pertence à camada de transporte e é similar ao TCP. Este protocolo é mais simples e mais rápido que o TCP, porém o que o faz ser menos confiável que este é o fato de não haver garantia e integridade dos pacotes. É bastante utilizado em serviços de *streaming* (Ex: YouTube[2]).

- ***Secure Shell* - SSH**

O protocolo SSH (*Secure Shell*) é um protocolo da camada de aplicação que tem como objetivo acessar um computador através de outro, remotamente, contanto que ambos estejam conectados numa rede insegura. Este protocolo utiliza criptografia de chave pública, e que fornece um canal seguro numa rede utilizando a arquitetura de cliente-servidor, já explicado anteriormente na sessão 2.1. Assim, um usuário de um sistema de computarores poderá fazer o login remoto, funcionando como um cliente-SSH, em outra máquina, que por sua vez funcionará como um servidor-SSH.

[2]https://youtube.com/

- ***Constrained Application Protocol* - CoAP**

Este protocolo é sem dúvida o mais importante para o entendimento deste trabalho. O protocolo CoAP (*Constrained Application Protocol*) é similar ao protocolo HTTP (*Hypertext Transfer Protocol*) que permite a transferência de arquivos via Web, porém este é voltado para dispositivos mais restritos, com algumas limitações.

De acordo com a RFC (*Request*) 7252: "O protocolo CoAP é um protocolo de transferência Web, especializado para uso de nós restritos e redes restritas na Internet das Coisas. O protocolo é projetado para aplicações machine-to-machine (M2M), como energia inteligente e automação predial"[7].

Assim como no HTTP, o protocolo CoAP utiliza operações de GET, POST, PUT e DELETE, e também realiza a descoberta de recursos Web através das URIs (*Uniform Resource Identifier*). Porém, o que torna o CoAP mais simples e limitado é o fato deste não utilizar o TCP para transmissão de pacotes, mas sim o UDP.

Nos pacotes UDP, o CoAP usa um cabeçalho binário de quatro bytes, seguido por uma sequência de opções. Esta codificação compacta, mas facilmente analisável, permite um tamanho total de cabeçalho de 10 a 20 bytes para uma solicitação típica. A codificação diferencial de tipos de opção fornece a extensibilidade futura necessária, sem sobrecarregar implementações simples [9].

2.2 Arcabouço da Open Connectivity Foundation

O arcabouço OCF é um conjunto de protocolos de comunicação e ferramentas que tiveram o seu uso integrado a fim de viabilizar um meio de comunicação para Internet das Coisas. Sua arquitetura permite interações baseadas em recursos entre artefatos de IoT, ou seja, dispositivos físicos ou aplicativos, aproveitando-se dos padrões e tecnologias existentes do setor e fornecendo soluções para estabelecer conexões (sem fio ou com fio). Além disso, permite gerenciar o fluxo de informações entre dispositivos, independentemente de seus fatores de forma, sistemas operacionais ou provedores de serviços [10].

2.2.1 Arcabouço OCF

A arquitetura do OCF é baseada na ROA (*Resource-oriented Architecture*), que utiliza suas operações aderidas ao REST (*Representational State Transfer*). REST é um estilo de arquitetura, ou seja, não é um conjunto específico de tecnologias, e sim um modelo que define um conjunto de restrições e propriedades baseados no protocolo HTTP. A ideia central do REST gira em torno da noção de recurso como qualquer componente de um aplicativo que precisa ser usado ou endereçado [11]. Tais recursos podem representar entidades do mundo físico, como sensor de temperatura, energia elétrica, dentre outros.

A arquitetura OCF é organizada principalmente através de três aspectos: modelo de recursos, operações RESTful e abstrações.

- **Modelo de Recurso**: o modelo de recurso fornece as abstrações e os conceitos necessários para modelar e operar logicamente na aplicação e em seu ambiente de uso. O modelo de recurso principal é propenso a se adaptar para qualquer domínio de aplicação específico. O modelo de recursos define um recurso que abstrai uma entidade e a representação de um recurso mapeia o estado da entidade.
- **Operações RESTful**: As operações genéricas CRUDN (*CREATE, RETRIEVE, UPDATE, DELETE e NOTIFY*), são definidas usando o paradigma RESTful para modelar as interações com um Recurso no modelo de uma tecnologia agnóstica, que permite que o usuário tenha o poder de decisão.
- **Abstração**: As abstrações no modelo de recursos e as operações RESTful são mapeadas para elementos concretos usando primitivas de abstração. Um manipulador de entidade é usado para mapear uma entidade para um recurso e primitivos de abstração de conectividade são usados para mapear operações RESTful lógicas para tecnologias ou protocolos de conectividade de dados.

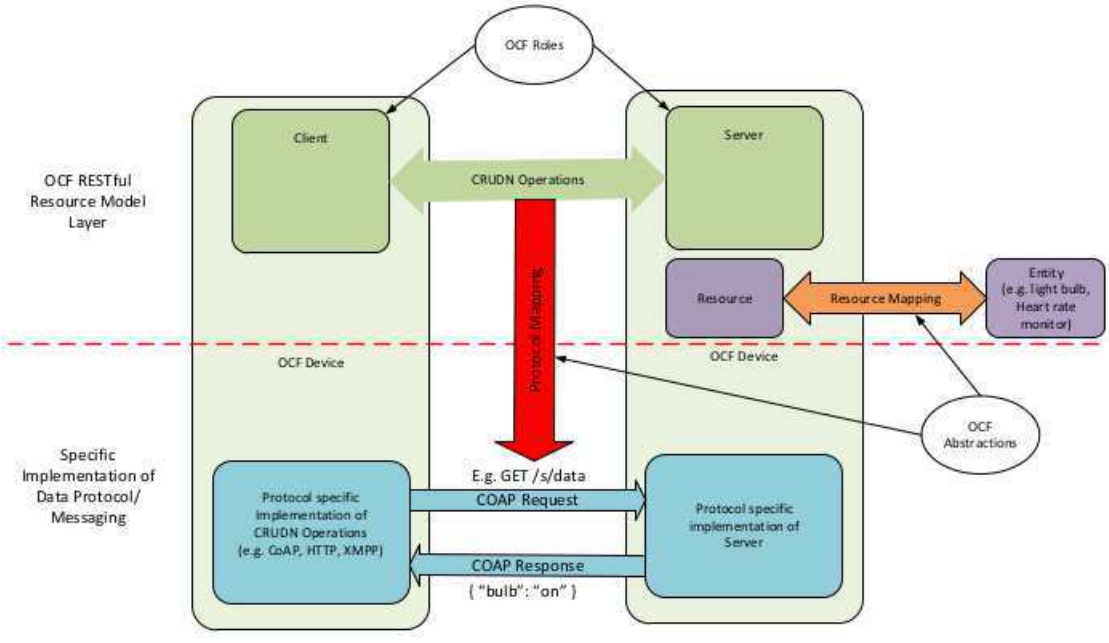

Figura 2.3: Arquitetura OCF (OCF Core Specification v2.0)

A estrutura do arcabouço pode ser descrita através do diagrama da Figura 2.3. O arcabouço do OCF é caracterizado por algumas funções que marcam as suas operações. Tais como:

- **Identificação e Endereçamento**: define o identificador e a capacidade de endereçamento.
- **Descoberta de dispositivos e recursos**: realiza a descoberta de dispositivos e recursos da rede.
- **Modelo de Recursos**: representa as entidades na forma de recursos e define mecanismo de manipulação desses recursos.
- **CRUDN**: provê um esquema de operações genéricas nas iterações entre cliente e servidor.
- **Mensagens**: provê protocolos de mensagem para uma operação RESTful (Ex: CoAP).
- **Gerenciamento de Dispositivos**: gerencia os recursos de um dispositivo e inclui a separação de dispositivo e configuração inicial, bem como monitoramento e diagnóstico de dispositivos.
- **Segurança**: fornece autenticação, autorização e controle de acesso às entidades.

2.2.2 IoTivity Framework

O IoTivity[3] é um framework de software de código aberto (*open source*) que permite a conectividade entre dispositivos a fim de atender as necessidades emergentes da IoT. O projeto IoTivity é uma implementação do arcabouço OCF, patrocinado pela Open Connectivity Foundation™, e foi criado para reunir a comunidade de código aberto para acelerar o desenvolvimento da estrutura e dos serviços necessários para conectar dispositivos na IoT [12].

O IoTivity possui uma versão simplificada de implementação do protocolo OCF conhecido como *IoTivity-Constrained*, que suporta a construção de aplicações seguras e interoperáveis para IoT [13]. Seu projeto de código aberto se encontra disponível na plataforma GitHub[4]. A Figura 2.4 mostra a estrututa da arquitetura do *framework*.

[3] iotivity.org
[4] https://github.com/

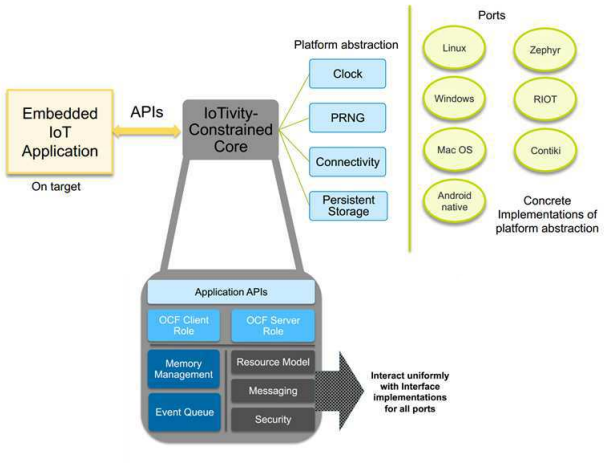

Figura 2.4: Arquitetura do IoTivity-Constrained

2.3 Hardware

Nesta seção serão detalhados os principais componentes e módulos de hardware utilizados para a implementação da solução proposta nesse trabalho.

2.3.1 Raspberry Pi Modelo B

O **Raspberry Pi** é um pequeno computador de placa única, desenvolvidos pela Raspberry Pi Foundation[5], que teve a sua primeira versão lançada em 2012. Na época de seu lançamento, obteve rápida popularidade (mais do que o esperado) por sua simplicidade e baixo custo. Possui como público-alvo estudantes de Ciências da Computação e Engenharia, que pretendem melhorar o aprendizado em tecnologias como ambientes inteligentes, robótica, e claro, Internet das Coisas.

A arquitetura dos modelos de Raspberry funcionam como um *System on a Chip* (SoC) que nada mais é do que ter praticamente todos os componentes necessários de um sistema computacional integrados em um único chip. Esses componentes naturalmente incluem uma CPU, memória, barramento de entrada e saída, etc.

Para a implementação do projeto foi utilizado uma das primeiras versões do Raspberry Pi: o **Pi Model B**. As suas especificações são as seguintes:

- SoC: Broadcom BCM2835 SoC;
- Processador: 700 MHz ARM1176JZF-S core CPU;
- Placa de Vídeo: Broadcom VideoCore IV GPU;

[5]https://www.raspberrypi.org

- Memória RAM: 512 MB;
- 2 entradas USB2.0;
- Saída de Vídeo via Composite (PAL e NTSC), HDMI ou Raw LCD (DSI);
- Sáida de Áudio via Jack 3.5mm ou por HDMI;
- Armazenamento: SD/MMC/SDIO;
- Entrada para conexão Ethernet 10/100 (RJ45)
- Periféricos de baixo nível: 8 pinos GPIO, UART, I2C bus, barramento SPI, +3.3V e +5V;
- Ground;
- Requisitos de Entrada: 5V @ 700 mA via MicroUSB ou cabeçalho GPIO;

Figura 2.5: Raspberry Pi Modelo B

2.3.2 Arduino Shield

O **Arduino Shield** é uma expansão do microcontrolador Arduino associada ao Rasberry Pi que permite o uso de qualquer um dos shields, placas e módulos projetados para o Arduino em um Raspberry Pi. Também inclui a possibilidade de conectar sensores digitais e analógicos, usando a mesma pinagem do Arduino, mas com a potência e os recursos do Raspberry [14].

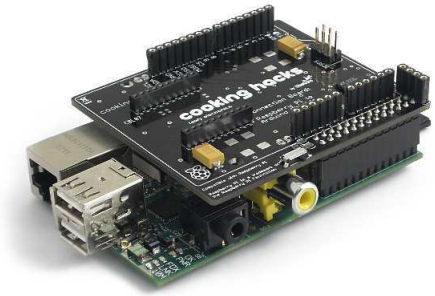

Figura 2.6: Arduino Shield

2.3.3 e-Health Sensor Shield

O **e-Health Sensor Shield** é uma placa de expansão para o *Arduino Shield* do Raspberry Pi e que permite que os usuários executem aplicações biométricas e médicas, onde o monitoramento do corpo é necessário usando diversos tipos de sensores. Essas informações podem ser usadas para monitorar em tempo real o estado de um paciente ou para obter dados confidenciais, a fim de serem posteriormente analisados para diagnóstico médico. As informações biométricas coletadas podem ser enviadas sem fio usando qualquer uma das seis opções de conectividade disponíveis: Wi-Fi, 3G, GPRS, Bluetooth, 802.15.4 e ZigBee, dependendo do aplicativo [15].

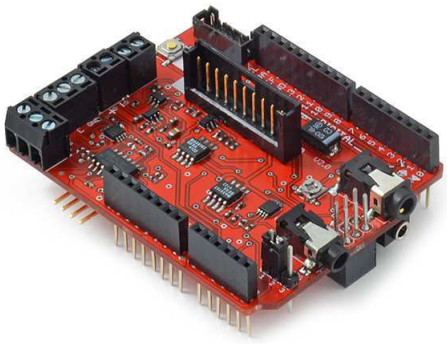

Figura 2.7: e-Health Sensor Shield para Raspberry

2.3.4 Oxímetro

O **oxímetro** é um aparelho que possui sensores ópticos que converte espectros de luz em sinais elétricos, sendo processados por um pequeno controlador. É um equipamento que funciona através de espectrofotometria que mede os batimentos cardíacos e nível de oximetria através da luz transmitida. Os sensores captam a luz que atravessa com o pulso cardíaco do dedo. A saturação e nível de oxigênio tem relação com a coloração do sangue. Então, quanto maior o brilho da cor vermelha, maior será a saturação. Através de um microcontrolador que efetua os cálculos dos sinais recebidos, este imprime na tela os valores de batimentos por minuto (bpm) e do nível de oxigênio no sangue (SpO_2).

Para que a leitura dos dados do oxímetro fosse passada para o Raspberry Pi, a ©Cooking Hacks projetou um sensor ligado ao dispositivo que se conecta à entrada GPIO do e-Health Sensor Shield, possibilitando assim, a comunicação entre ambos.

Figura 2.8: Oxímetro para plataforma e-Health

3 Proposta e Solução

Nesta seção será detalhada a proposta de solução ao problema do trabalho. A solução incluirá a arquitetura da aplicação, a estrutura da comunicação cliente-servidor, bem como os seus respectivos diagramas de estado.

3.1 Descrição da Solução

Avanços na informação, telecomunicações e redes de tecnologia desempenham um papel significativo nos sistemas de saúde e têm contribuído baastante no desenvolvimento de informações médicas sistemas. No entanto, os cuidados de saúde representam um dos maiores desafios socioeconômicos que cada país enfrenta [16]. Os problemas relacionados à saúde atingem principalmente as pessoas com baixa rentabilidade financeira, devido aos altos custos que alguns tratamentos médicos requerem.

O desenvolvimento de sistemas *healthcare* exige um esforço concentrado para aproveitar o poder da informação e tecnologias de comunicação ao serviço da saúde a fim de criar dados mais eficientes, eficazes e seguros para o compartilhamento, o processamento de informações de saúde em grande escala, dentre outros fatores [17].

A solução proposta de baixo custo desse trabalho, que corresponde ao cenário do *healthcare*, é a implementação de um dispositivo oxímetro interoperável para Internet das Coisas, através de uma relação cliente-servidor entre o dispositivo, chamado de Servidor-Oxímetro e um software cliente, chamado de Cliente-Saúde.

O oxímetro irá atuar acoplado com um Raspberry Pi Modelo B (ver Figura 2.4) através de dois shields de automação (*e-Health* e Arduino) e servirá como o servidor do sistema. Este servidor coletará os dados obtidos pela leitura do oxímetro e os preparará para o envio caso algum *host* conectado na rede, no caso o Cliente-Saúde, realize a requisição dos dados obtidos pelo oxímetro.

3.2 Estrutura de Comunicação

A estrutura de comunicação entre o Servidor-Oxímetro e o Cliente-Saúde se dá através do protocolo OCF. Esse arcabouço se destaca pela sua arquitetura REST, que por sua vez é caracterizado pelo seu desempenho rápido, sua confiabilidade e escalabilidade, por meio da reutilização de componentes que podem ser gerenciados e atualizados sem afetar o sistema como um todo, mesmo que esteja em execução.

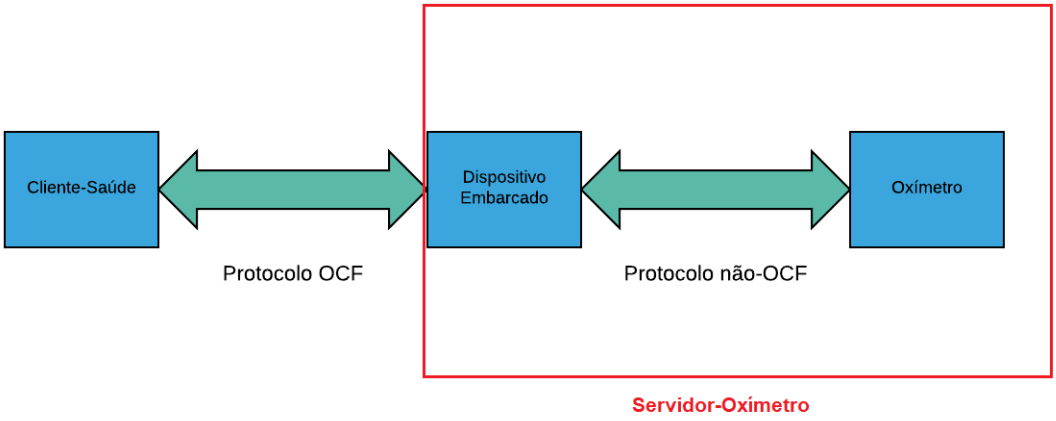

Figura 3.1: Servidor-Oxímetro em ponte com dispositivo oxímetro

O modelo base arquitetural da solução é apresentado através da ilustração da Figura 3.1. Nele, ocorre que o oxímetro realiza a leitura de dados e os encaminha para o Servidor-Oxímetro, que por sua vez enviará esses dados para um Cliente-Saúde, quando solicitado. O princípio de funcionamento desse sistema ocorre internamente através dos ***handlers***, ou manipuladores de entidade. Ao inicializar, o servidor executa esses *handlers*, que descobrem os sistemas não-OCF (no caso o oxímetro) e cria recursos para cada dispositivo ou funcionalidade descobertos. O *handler* de entidade cria um recurso para cada dispositivo ou funcionalidade descoberta e conecta-se a esse recurso. Esses recursos são disponibilizados pelo servidor à rede de comunicação.

Depois que os recursos forem criados e tornam-se indentificáveis, o dispositivo de exibição, no caso o Cliente-Saúde, poderá descobrir esses recursos e operá-los. As solicitações para um recurso no servidor são então interpretadas pelo *handler* de entidade e encaminhadas para o oxímetro usando o protocolo suportado pelo mesmo. As informações retornadas do oxímetro são então mapeadas para a resposta apropriada para esse recurso.

3.2.1 Modelo de Recursos

O Modelo de Recursos define conceitos e mecanismos que fornecem consistência e interoperabilidade básica entre dispositivos nos ambientes OCF. Os conceitos e mecanismos deste modelo são mapeados para os protocolos de transporte para permitir a comunicação entre os dispositivos. Cada transporte fornece a interoperabilidade do protocolo de comunicação. O Modelo de Recursos, portanto, permite que a interoperabilidade seja definida independentemente do método de transporte.

Os recursos são definidos usando tipos de dados derivados do modelo JSON (*JavaScript Object Notation*), conforme definido na RFC 7159 [18]. O JSON é um formato para troca de dados e informações, independentemente das linguagens de programação dos sistemas envolvidos, derivado da linguagem JavaScript. No entanto, um recurso pode sobrecarregar um valor definido de

JSON para especificar um subconjunto específico do mesmo, usando palavras-chave de validação definidas na no esquema de validação JSON.

O modelo JSON do oxímetro possui como recursos, ou seja, os dados a serem trocados, os valores de batimento por minuto (bpm) e do nível de oxigênio do sangue em porcentagem (SpO_2). Ambos representam variáveis do tipo inteiro. Esses recursos são definidos durante a inicialização do sistema junto ao Servidor-Oxímetro, que por sua vez serão descobertos por um *handler* presente no Cliente-Saúde. A Figura 3.2, apresenta uma representação do esquema de modelo JSON da aplicação.

```
{
"rt": ["core.oximeter", "core.oxygen"],
"id": "/a/oximeter",
"spo2":{"type": "integer", "minimum": 0, "maximum": 100},
"BPM":{"type": "integer", "minimum": 0}
}
```

Figura 3.2: Esquema de modelo JSON da aplicação do oxímetro

3.3 Arquitetura da Aplicação

Nesta subseção serão apresentados detalhes das arquiteturas do Servidor-Oxímetro e Cliente-Saúde para a aplicação de solução do problema proposto, bem como os detalhes das transições dos estados de operação.

3.3.1 Servidor-Oxímetro

O **Servidor-Oxímetro** é o principal responsável pelo encaminhamento de dados para um cliente que deseja realizar a leitura dos dados obtidos pelo oxímetro. Ele atua paralelamente com a leitura dos dados do dispositivo independentemente da requisição ou não de dados realizadas pelo cliente. A forma como esse processo de leitura de dados é realizada já foi apresentada anteriormente na Figura 3.1.

O seu modo de operação funciona da seguinte forma: assim que o dispositivo oxímetro se conecta ao servidor, este irá esperar pela conexão de um cliente, ambos presentes em uma mesma rede LAN. A partir do momento em que o cliente se conecta, o servidor começa a realizar a leitura de dados do dispositivo, ou seja, ele recolhe os valores de bpm e SpO_2 e os transforma em recursos através dos *handlers*.

Essa coleta de dados é realizada periodicamente, com os valores sendo atualizados em um intervalo de 1s. Em paralelo à leitura de dados, o servidor espera pela requisição do cliente: caso

ele receba, os valores lidos no momento da requisição são enviados ao cliente, caso contrário, continuará a realizar as devidas leituras até a chegada de uma nova requisição. O princípio de funcionamento pode ser descrito através do diagrama de estados apresentado na Figura 3.3 abaixo.

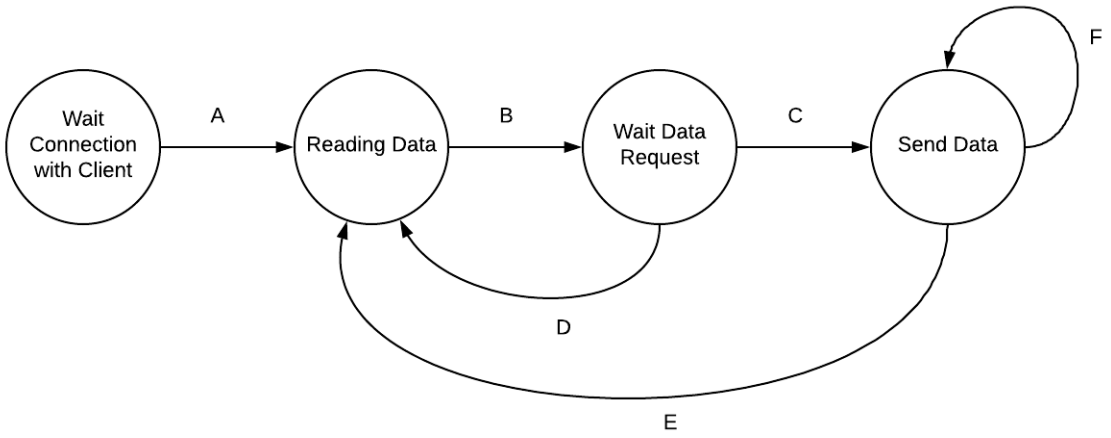

Figura 3.3: Diagrama de Estados do Servidor-Oxímetro

As letras do diagrama acima representam as transições entre os estados de operação do Servidor-Oxímetro. Onde:

- **A**: Conexão estabelecida. Após isso, o oxímetro passa a realizar a leitura dos dados;
- **B**: Leitura de dados realizada. O servidor do oxímetro irá esperar a requisição de dados do cliente;
- **C**: Requisição de dados recebida. O servidor irá enviar os dados para o cliente;
- **D**: Timeout. O cliente não fez a requisição dos dados, durante um intervalo de aproximadamente 1 segundo, o servidor realizará uma nova leitura de dados;
- **E**: Dados enviados para o cliente. Após isso, o servidor retorna à leitura de novos dados;
- **F**: Houve falha no envio dos dados. O servidor retorna a tentar realizar o envio.

3.3.2 Cliente-Saúde

O **Cliente-Saúde** é o que possibilita um *host* remoto obter acesso aos dados lidos pelo oxímetro e enviados através do Servidor-Oxímetro. Seu funcionamento acontece por meio de uma conexão em rede LAN com o servidor. Ao estabelecer uma conexão com o servidor, o Cliente-Saúde irá realizar o *discovery* (descoberta) dos recursos criados pelos *handlers* do servidor. Essa descoberta também é realizada através de *handlers* fornecidos pelo arcabouço OCF.

Assim que descobertos os recursos, o cliente estará apto para realizar a requisição de dados e obter seus devidos valores. O diagrama de estados do mesmo se encontra na Figura 3.4 logo abaixo.

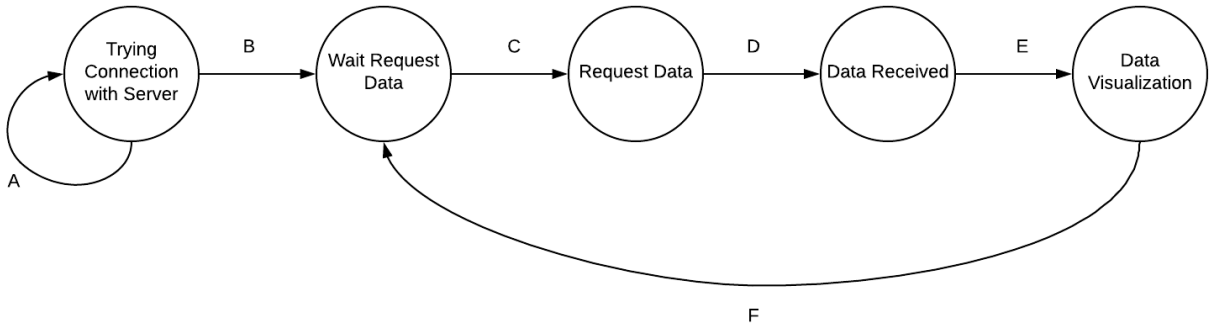

Figura 3.4: Diagrama de Estados do Cliente-Saúde

Assim como no Servidor-Oxímetro, as letras presentes no diagrama Cliente-Saúde apresentado acima, representam as transições entre os seus estados de operação. Onde:

- **A**: O cliente tenta se conectar com o servidor. Caso não consiga, ele irá tentar realizar a conexão novamente até obter sucesso ou ocorrer timeout;
- **B**: Conexão estabelecida. Após isso, o cliente entra em estado de espera até este decidir fazer uma requisição dos dados do servidor;
- **C**: O cliente então realiza a requisição de dados;
- **D**: A requisição de dados é feita. O cliente então recebe os dados obtidos pelo servidor;
- **E**: Dados recebidos. Após isso, o cliente então visualiza os dados recebidos;
- **F**: Dados visualizados. Após isso, o cliente então entra novamente em estado de espera até fazer a requisição de novos dados.

4 Desenvolvimento

Nesta seção apresentados detalhes a forma de como foi implementado o projeto de aplicação à solução, assim como os devidos fatores que comprovam e validam a sua legitimidade.

Como já constatado na seção anterior, o projeto constitui de uma aplicação de um dispostivo oxímetro para Internet das Coisas. Sendo assim, iremos abordar o modelo de relação cliente-servidor elaborado para estabelecer a comunicação desejada, que por fim será responsável pela troca de informações entre eles através dos recursos pré-definidos no modelo JSON.

4.1 Implementação do Servidor-Oxímetro

Para dar início ao desenvolvimento de um dispositivo que fosse capaz de atuar como servidor na aplicação, foi utilizado um projeto de referência da ©Cooking Hacks utilizando o *e-Health Sensor Shield* como plataforma para sensores de dispositivos da biomedicina[6]. Logo, utilizou-se esta plataforma junto ao sensor de oximetria, o oxímetro. Para tal, foi escolhido o uso de um Raspberry Pi Modelo B, que por sua vez se conectava junto a uma expansão para um Arduino, a fim de poder se comunicar com o oxímetro através da plataforma *e-Health*.

Este projeto inclui um código na linguagem de programação C++ que oferece uma biblioteca de referência a qual permite realizar a leitura dos valores coletados pelo oxímetro em um Raspberry Pi. O sensor de oxímetro (ver Figura 2.8) possui uma entrada para GPIO, que é conectada à plataforma *e-Health*. Através dela, ocorre uma interrupção em um dos pinos, para que se possa coletar dados do sensor. Assim que realizada a interrupção, uma função de leitura do sensor é chamada e os valores de bpm e SpO_2 são coletados.

Para que esses dados possam ser visualizados pelo cliente, no caso o usuário da aplicação, é preciso primeiro estabelecer uma conexão entre o Raspberry Pi e o *host* do cliente. Para isso, utilizou-se as ferramentas do software IoTivity, que pudesse utilizar o OCF como ponte para a comunicação entre o Servidor-Oxímetro e o Cliente-Saúde.

Utilizando as funções fornecidas pelo IoTivity, foi possível realizar o registro dos recursos de entidades do dispositivo oxímetro, de acordo com o seu modelo JSON, assim como determinar a execução dos *handlers* para criá-los quando o servidor-iotivity é inicializado.

Após a criação dos recursos, o servidor-iotivity entra em um *loop* para a execução de tarefas em paralelo necessárias para o funcionamento do arcabouço OCF.

Toda vez que um novo evento acontece, uma nova *thread* é iniciada para o processamento do mesmo. Esses eventos, nada mais são do que a captura dos valores lidos pelo oxímetro. Essa coleta é feita através de uma função *get_oximeter*, que é executada toda vez que o cliente realiza

[6]https://www.cooking-hacks.com/documentation/tutorials/ehealth-v1-biometric-sensor-platform-arduino-raspberry-pi-medical

um *request* ao servidor-iotivity, requisitando os valores de leitura do oxímetro. Ao mesmo tempo em que lê os valores do oxímetro, a função envia tais dados para o cliente, quando requisitado. Ao final da requisição, a *thread* é desencadeada e o ciclo se repete até que o processo se encerre.

Para realizar essa captura de valores do oxímetro pelo servidor, foi necessário realizar a comunicação entre os processos de leitura de oximetria e de comunicação IoTivity. Essa troca de informações entre ambos, é conhecido como **IPC**, ou **Inter-Process Communication**. Para isso foi desenvolvido um modelo de comunicação IPC por meio de um soquete (do inglês, *socket*), que permite o acesso a um ponto de comunicação entre os dois processos para que haja a leitura e escrita de dados entre eles. Este método foi utilizado devido a restrições de tempo e pela complexidade de se implementar ambos os códigos em conjunto, visto que o código de leitura de dados do oxímetro, disponibilizado pelo ©Cooking Hacks, utiliza bibliotecas em linguagem C++, enquanto que o servidor do IoTivity possui linguagem C.

Logo, toda vez que o programa de leitura do oxímetro é executado, uma porta do Raspberry Pi se abre, realizando o chamado *listening*, e espera o servidor-iotivity realizar contato através da porta.

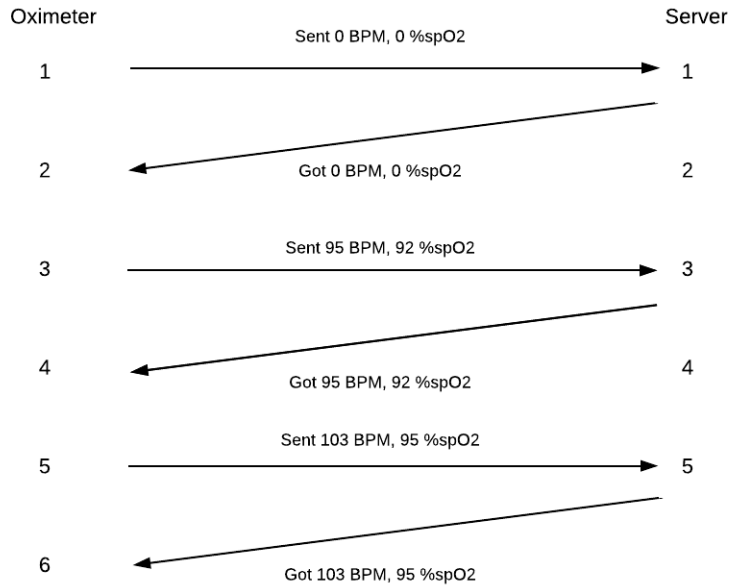

Figura 4.1: Diagrama IPC do Servidor-Oxímetro

A Figura 4.1 apresenta um diagrama de sequência com um exemplo de como funciona a comunicação interprocessos (IPC) entre o oxímetro e o servidor-iotivity. Após estabelecerem a conexão com o soquete, o servidor-iotivity passa a receber os dados das leituras do oxímetro. Inicialmente se recebem valores iniciais, mas posteriormente os dados são coletados normalmente. Ao

receber os dados do oxímetro, o servidor-iotivity espera a requisição do Cliente-Saúde para enviá-los para o usuário. Após a requisição, ele os envia de volta ao dispositivo para receber um novo conjunto de dados atualizado. Parte dessa comunicação também pode ser visualizada pela Figura 3.3.

A montagem completa do Servidor-Oxímetro pode ser conferida na Figura 4.2 abaixo.

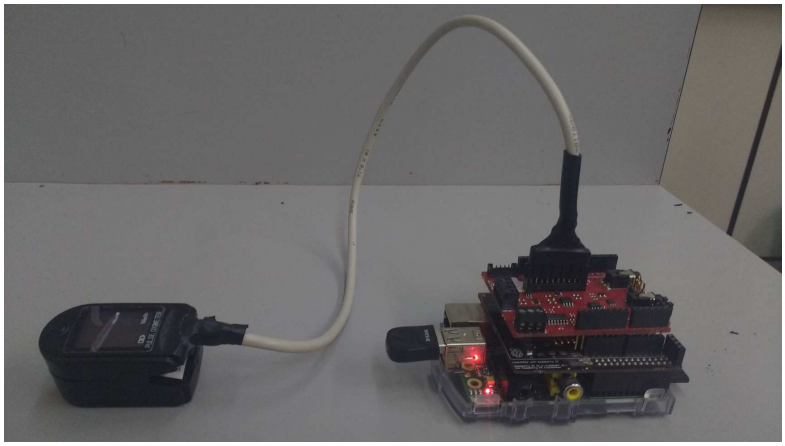

Figura 4.2: Servidor-Oxímetro devidamente montado

4.2 Implementação do Cliente-Saúde

O Cliente-Saúde foi implementado de forma semelhante ao servidor, pois também utiliza as ferramentas do software IoTivity como a implementação do arcabouço OCF. A principal diferença entre o código do cliente para o código do servidor-iotivity foi que ao invés de realizar o registro dos recursos do oxímetro, o cliente-iotivity irá realizar a descoberta desses recursos já criados no servidor-iotivity.

Ao iniciar o processo, o cliente-iotivity executará os *handlers*, necessários para a realização da descoberta de recursos. em seguida, este entra em um *loop*, semelhante ao loop do servidor-iotivity, onde uma *thread* é encadeada para a ocorrência de eventos na aplicação.

Os eventos iniciais do processo são reservados para a descoberta dos recursos. Assim que descobertos os recursos, os novos eventos do *loop* são passados para a requisição de dados. Semelhante ao Servidor-Oxímetro, no Cliente-Saúde também foi criada uma função chamada de *get_oximeter*, porém ao invés de esperar a requisição, ela será responsável por realizá-la.

A cada *request* feito pelo cliente, caso o servidor esteja executando dentro da mesma rede LAN, os dados lidos pelo oxímetro serão transmitidos para um usuário em tempo-real, de modo que este possa visualizar os valores de bpm e SpO_2 através da tela do seu computador ou *smartphone*, dependendo de qual tipo de *host* ele esteja utilizando.

Essas requisições ocorrem de modo contínuo, até que ocorra um erro na conexão ou quando um

dos processos é encerrado, seja ele o servidor ou o cliente.

Desta forma, buscou-se a criação de uma interface gráfica que coletasse os valores dos dados do oxímetro, assim que realizada a requisição de forma manual. A interface gráfica foi desenvolvida utilizando a biblioteca *tkinter*, da linguagem Python. Novamente, para que os dados do cliente-iotivity pudessem ser acessados pela interface, foi utilizado um soquete para comunicação entre ambos os programas, estabelecendo assim uma IPC. O comportamento do IPC entre o cliente-iotivity e sua interface gráfica é similar ao do servidor-iotivity com o dispositivo oxímetro. O diagrama de sequência da comunicação entre ambos pode ser vista na Figura 4.3.

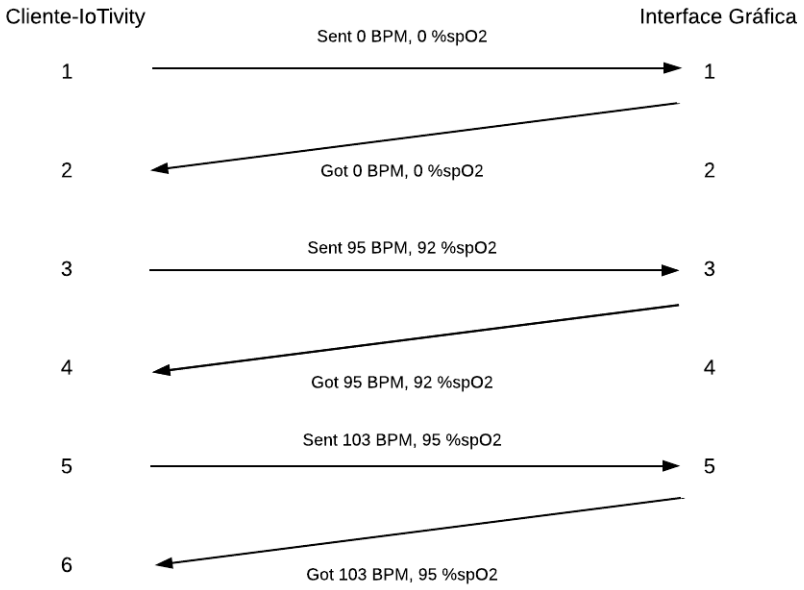

Figura 4.3: Diagrama IPC do Cliente-Saúde

Ao executar o programa do cliente-iotivity, este irá escutar uma determinada porta no endereço IP local da máquina (*localhost*), esperando uma resposta da interface gráfica. Assim, a interface é inicializada e abre uma conexão com o cliente-iotivity, que posteriormente irá realizar a descoberta de recursos e preparar o ambiente para as devidas requisições poderem ser feitas pelo usuário, através da interface gráfica. Esse conjunto de programas,interface gráfica e cliente-iotivity, que se comunicam um com o outro formam o Cliente-Saúde. A janela da interface gráfica criada para o cliente-iotivity pode ser visualizada na Figura 4.4 abaixo.

Figura 4.4: Interface Gráfica para o Cliente-Saúde

5 Validação

Esta seção foi reservada para o detalhamento dos testes realizados para a validação das implementações feitas para o Servidor-Oxímetro e o Cliente-Saúde.

5.1 Validação do Servidor-Oxímetro

Ao finalizar a implementação do Servidor-Oxímetro, o mecanismo estava pronto para a realização de uma série de testes. Com o propósito de validar apenas o Servidor-Oxímetro, ou seja, garantir que ele esteja retornando os resultados de acordo com o esperado, foi necessária a utilização de um cliente de testes para realizar as devidas requisições ao servidor.

Com o propósito de realizar requisições OCF com um cliente de testes, foi utilizado um cliente OCF disponibilizado pelo IoTivity. A ferramenta escolhida para tal objetivo foi o Eclipse[7]. O Eclipse é um ambiente de desenvolvimento integrado, o qual permite desde o desenvolvimento de software para diversas linguagens, até a execução de ferramentas integradas através de plugins. No nosso caso iremos utilizá-lo como um cliente do OCF, isto é, um cliente genérico com propósito geral. Para isso, foi necessário instalar um *plugin* que executa um cliente IoTivity integrado ao Eclipse, de modo que é possível utilizar o Eclipse para a descoberta de recursos e requisição de dados via protocolo OCF, como por exemplo, o dispositivo oxímetro desenvolvido[8].

Ao instalar o plugin, o software do Eclipse estava pronto para funcionar como cliente da nossa aplicação. Para os testes, foi desenvolvido um script Linux na plataforma embarcada do Raspberry Pi, o qual realiza a inicialização automática do servidor-OCF e dos módulos de coleta de dados de oximetria. Após essa inicialização, o servidor-OCF fica passível de ser descoberto via clientes OCF.

Com o Servidor-Oxímetro ativo, no Eclipse, aciona-se o botão de *Find Resources*, ou ”encontrar recursos”, que irá descobrir os recursos fornecidos pelos *handlers* do servidor. Esta função de captura de recursos representa o *discovery* do Cliente-Saúde. Os recursos foram encontrados com sucesso, bem como seus atributos (bpm e SpO_2), que são atualizados na medida que o GET (requisição) é acionado, fazendo com que assim, os valores lidos pelo oxímetro sejam visualizados em tempo-real pela janela do Eclipse.

[7] https://www.eclipse.org
[8] https://wiki.iotivity.org/iotivity_tool_guide

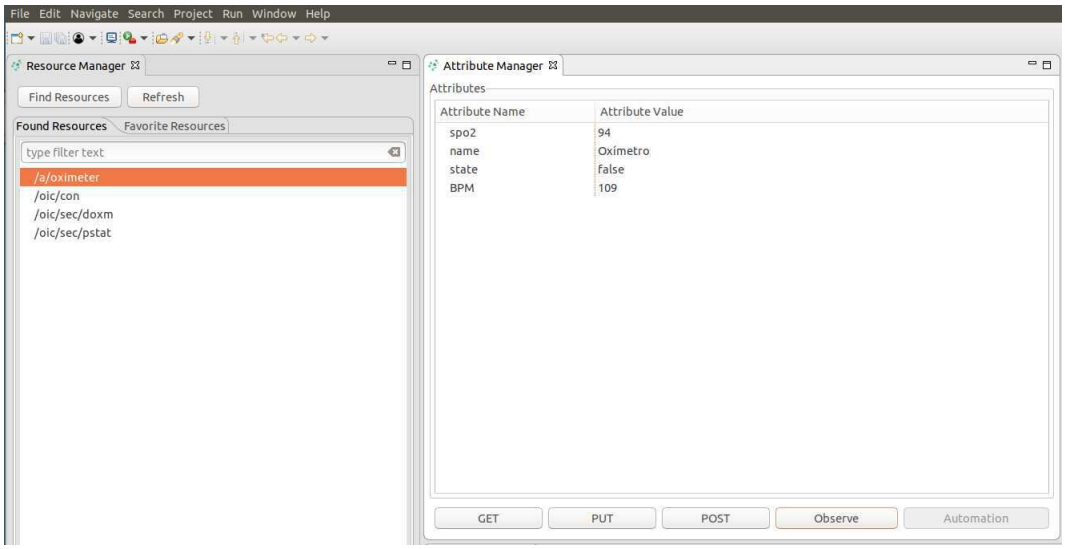

Figura 5.1: Teste de validação do Servidor-Oxímetro usando o Eclipse

Como pode ser visto pela Figura 5.1, o Servidor-Oxímetro funciona de acordo com o esperado. Os valores de bpm são coletados corretamente, porém foi identificado que eventualmente os valores de oximetria retornam dados errôneos, onde muitas vezes apenas o valores da casa das unidades eram lidos corretamente, enquanto que a casa das dezenas apresentava instabilidade em seus valores. Um possível motivo para tal problema pode ser por causa de um pino defeituoso na entrada GPIO do *shield*. Os procedimentos gerais para esse teste de validação pode ser visto com mais detalhes na tabela da Figura 5.2.

Após a confirmação de sua legitimidade, um *script* foi elaborado para que os programas do Servidor-Oxímetro executassem automaticamente toda vez que o Raspberry Pi é inicializado, tornando os testes futuros bem mais práticos desta forma, assim como o uso da aplicação mais fácil para os usuários.

Tipo de Teste	Condições para Teste	Resultados	Status do Resultado
Conexão de oxímetro e servidor-iotivity	Raspberry Pi conectado à uma rede interna. São necessários dois terminais de linhas de comando para a execução dos arquivos.Um para o oxímetro e outro para o servidor-iotivity. Executar primeiro o programa do oxímetro e em seguida, executar o servidor-iotivity.	Ambos os processos se comunicam perfeitamente através da porta, contanto que ela não esteja sendo utilizada em um outro processo já existente.	Ótimo
Registro de recursos	Enquanto os processos dos programas do Servidor-Oxímetro estiverem rodando no Raspberry Pi, inicializa-se o software Eclipse em uma mpaquina que esteja conectada na mesma rede do mesmo. Selecionar a opção de descoberta de recursos.	Os recursos criados pelos handlers do servidor-iotivity foram encontrados com sucesso pelo simulador de cliente do Eclipse.	Ótimo
Coleta de dados	Selecionar o recurso identificado que representa o dispositivo de oximetria. Realizar um GET para coletar dados do dispositivo.	O simulador recebe os dados enviados pelo oxímetro. Os valores do batimento cardíaco são coletados corretamente e em tempo real, porém os valores de oximetria muitas vezes não retornam o valor correto.	Bom, porém apresenta falhas

Figura 5.2: Tabela de Procedimentos para os testes de validação do Servidor-Oxímetro

5.2 Validação do Cliente-Saúde

Com o teste do Servidor-Oxímetro sendo bem-sucedido, um cliente-iotivity agora poderá utilizá-lo como base para testar sua vericidade na aplicação. Inicialmente, o teste foi feito utilizando o *gnome-terminal* de uma máquina Linux, usada como o *host* do cliente-iotivity, para a visualização dos valores coletados.

Assim como no Eclipse, o cliente-iotivity realiza a descoberta dos recursos para então poder requisitar os dados do Servidor-Oxímetro. Porém, diferentemente da ferramenta de plugin do Eclipse, essas requisições são feitas automaticamente, já que enquanto um processo está sendo executado em um terminal de linhas de comando, nenhum outro comando pode ser proferido até que este processado seja morto, ou interrompido. Essa série de requisições automáticas ocorre por causa do *loop* gerado para a criação de *threads* no sistema, que realizam requisições infinitamente até a ocorrência de um erro na conexão.

Ao executar o arquivo relacionado ao cliente-iotivity, o mesmo irá tentar realizar a descoberta dos recursos. Porém, em várias ocasiões ocorre que o programa se encontra preso em um *loop*, devido a execução de uma *thread* responsável por realizar a tarefa relacionada a função que realiza a descoberta dos recursos. Logo, por muitas vezes o programa precisou ser reiniciado até que a descoberta fosse feita com êxito. Uma tabela de testes de validação para o cliente-iotivity se encontra na Figura 5.3 abaixo.

Tipo de Teste	Condições para Teste	Resultados	Status do Resultado
Conexão de Servidor-Oxímetro e cliente-iotivity	Raspberry Pi conectado à uma rede interna. Os processos relacionados ao Servidor-Oxímetro precisam estar rodando assim que o dispositivo é inicializado. Executar programa relacionado ao cliente-iotivity.	Ambos os processos se comunicam perfeitamente através do arcabouço OCF.	Ótimo
Descoberta de recursos	Ao inicializar o cliente-iotivity, realizar uma depuração no código e verifica-se se os recursos criados pelo servidor-iotivity estão sendo descobertos.	Os recursos são descobertos algumas vezes. Na maioria dos casos o programa fica preso em um loop causado pela execução de uma thread, não realizando a descoberta dos recursos.	Razoável
Coleta de dados	Após a descoberta, os valores do oxímetro são coletados de forma automática pelo processo em execução.	O simulador recebe os dados enviados pelo oxímetro. Os valores do batimento cardíaco são coletados corretamente e em tempo real, porém os valores de oximetria muitas vezes não retornam o valor correto.	Bom, porém apresenta falhas

Figura 5.3: Tabela de Procedimentos para os testes de validação do cliente-iotivity

Devido a esse fator, o usuário fica impossibilitado de realizar requisições manualmente. Além disso, não é nada prático ter uma aplicação biomédica, que só seja possível de se executar através de um terminal de linhas de comando, visto que uma pessoa comum, ou até um profissional da área de saúde, normalmente não tem conhecimentos necessários para utilizar tal ferramenta de forma concreta.

Como explicado na seção anterior, uma interface gráfica foi criada para solucionar esse problema. Na Figura 4.4, pode-se visualizar melhor o formato de requisição da interface. Assim que conectado com o Servidor-Oxímetro, o usuário pode utilizar a interface para realizar a captura de dados do dispositivo. De forma similar à simulação feita no Eclipse para o servidor do sistema, o Cliente-Saúde foi capaz de estabelecer uma troca de informações legítima com o mesmo. A tabela relacionada aos testes de validação para a interface gráfica do Cliente-Saúde podem ser visualizada na figura 5.4 abaixo.

Tipo de Teste	Condições para Teste	Resultados	Status do Resultado
Conexão do cliente-iotivity com a interface	A máquina que contém ambos os arquivos precisa estar conectada à rede interna. Executa-se primeiro o cliente-iotivity e em seguida a interface para haver comunicação interprocessos.	Ambos os processos se comunicam perfeitamente através de uma porta.	Ótimo
Descoberta de recursos	Ao inicializar o cliente-iotivity, realizar uma depuração no código e verifica-se se os recursos criados pelo servidor-iotivity estão sendo descobertos.	Os recursos são descobertos algumas vezes. Na maioria dos casos o programa fica preso em um loop causado pela execução de uma thread, não realizando a descoberta dos recursos.	Razoável
Coleta de dados	Após a descoberta, os valores do oxímetro são coletados de forma manual através de um botão da interface gráfica que realiza a requisição ao Servidor-Oxímetro.	O simulador recebe os dados enviados pelo oxímetro. Os valores do batimento cardíaco são coletados corretamente e em tempo real, porém os valores de oximetria muitas vezes não retornam o valor correto. Após algumas coletas de dados, a interface gráfica para de responder.	Bom, porém apresenta falhas

Figura 5.4: Tabela de Procedimentos para os testes de validação da interface gráfica

6 Conclusão

A Internet das Coisas é um conceito que pouco a pouco vem ganhando espaço no cenário industrial, onde existe a premissa de ser o estopim para a 4ª Revolução Industrial. Além disso, esse ramo da tecnologia também promete atingir setores de mais contato com o mundo civil, a exemplo de hospitais e centros médicos. Sua composição ao lado do estudo da engenharia biomédica requer uma certa atenção maior a ponto de agilizar e melhorar o atendimento das pessoas, tudo isso a baixo custo.

Vale ressaltar também a importância que o arcabouço OCF pode representar em projetos futuros, não só para o setor da biomedicina, mas também em áreas onde existe a possibilidade de se empregar o conceito de IoT a ponto de melhorar e otimizar o dia-a-dia de seus usuários.

A proposta deste trabalho cumpriu boa parte de suas expectativas, no que diz respeito à conexão entre dois dispositivos em uma mesma rede, que realizam a troca de informações obtidas pelo oxímetro. Informações essas representadas por recursos definidos por um modelo JSON e criados pelo OCF. A comunicação entre o oxímetro e o usuário, pôde ser feita através de uma relação cliente-servidor, com o auxílio de um dispositivo embarcado (Raspberry Pi) e dos protocolos que compõem o arcabouço OCF.

Foi possível a criação de uma interface gráfica para que o usuário pudesse usufruir do aplicativo de forma mais prática e fácil, visto que nem todos aqueles que fazem parte do público-alvo do projeto sabem como utilizar um terminal de linhas de comando em um sistema operacional UNIX, por exemplo.

No entanto, alguns problemas não puderam ser resolvidos, como no caso na coleta de dados de oximetria feita de forma errônea pelo *shield* disponibilizado pela ©Cooking Hacks. Enquanto que os batimentos cardíacos eram lidos de forma correta, a oximetria apresentava instabilidade na leitura pelos *shields*, diferentemente do que ocorria no dispositivo real do oxímetro. Tal problema começou a ocorrer ao final do desenvolvimento, ao contrário do que ocorria no início, no qual o *shield* enviava os dados de forma correta.

Outra questão a ser resolvida refere-se à descoberta de recursos realizada pelo cliente-iotivity. É possível que se manipule o código de execução da *thread* responsável pela tarefa, para que este retorne um *timeout* após determinado tempo caso não seja possível realizar a descoberta dos recursos do oxímetro. Porém, devido à restrições de tempo para o desenvolvimento deste trabalho, não foi possível relatar isto nesse trabalho.

Por último, é possível que em trabalhos futuros melhore-se alguns fatores, como criar janelas múltiplas para a interface gráfica, onde é exibido ao usuário cada evento que está ocorrendo durante a execução do Cliente-Saúde, separados em cada janela. Sugere-se também, a implementação da interface gráfica para plataformas Android, por exemplo, na qual facilitaria ainda mais o uso da

aplicação por profissionais. Também é interessante avaliar a possibilidade de evitar realizar uma comunicação de interprocessos entre os programas de ambos o Servidor-Oxímetro e Cliente-Saúde. Devido a uma certa complexidade que existe nessas atividades citadas, também não foi possível relatá-las no projeto.

Outro ponto interessante, que pode ser melhorado, trata-se da questão relacionada à segurança da aplicação, uma vez que é importante haver um sigilo para com os dados que estão sendo coletados. Além disso, o IoTivity também fornece uma funcionabilidade para a autenticação do sistema.

Referências

[1] WETHERALL, J.; TANENBAUM, A. S. Redes de Computadores. 5ª edição. *Rio de Janeiro: Editora Campus*, 2011.

[2] DACOSTA, Francis. Rethinking Internet of Things *A Scalable Approach to Connecting Everything* (2013).

[3] CATARINUCCI, Luca et al. An IoT-aware architecture for smart healthcare systems. *IEEE INTERNET OF THINGS JOURNAL*, v. 2, n. 6, p. 515-526, 2015.

[4] BELLO, Oladayo; ZEADALLY, Sherali; BADRA, Mohamad. Network layer inter-operation of Device-to-Device communication technologies in Internet of Things (IoT). *Ad Hoc Networks* v. 57, p. 52-62, 2017.

[5] https://openconnectivity.org/ - *Acessado em 23 de setembro de 2018*.

[6] KUROSE, James F.; ROSS, Keith W. *Redes de computadores: uma abordagem top-down*. 6ª Edição, Pearson, 2013.

[7] https://tools.ietf.org/html/rfc7252 - *Acessado em 06 de dezembro de 2018*.

[8] MOTA FILHO, João Eriberto. *Análise de tráfego em Redes TCP/IP*. São Paulo: Novatec Editora, 2013.

[9] BORMANN, Carsten; CASTELLANI, Angelo P.; SHELBY, Zach. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, v. 16, n. 2, p. 62-67, 2012.

[10] https://openconnectivity.org/specs/OCF_Core_Specification_v2.0.0.pdf - *Acessado em 19 de novembro de 2018*.

[11] GUINARD, Dominique; TRIFA, Vlad; WILDE, Erik. A resource oriented architecture for the web of things. In: *Internet of Things (IOT), 2010*. IEEE, 2010. p. 1-8.

[12] https://iotivity.org/about - *Acessado em 17 de novembro de 2018*.

[13] https://github.com/iotivity/iotivity-constrained - *Acessado em 19 de novembro de 2018*.

[14] https://www.cooking-hacks.com/raspberry-pi-to-arduino-shield-connection-bridge - *Acessado em 12 de novembro de 2018*.

[15] https://www.cooking-hacks.com/ehealth-sensor-shield-biometric-medical-arduino-raspberry-pi - *Acessado em 12 de novembro de 2018*.

[16] CHIUCHISAN, Iuliana; COSTIN, Hariton-Nicolae; GEMAN, Oana. Adopting the internet of things technologies in health care systems. In: *Electrical and Power Engineering (EPE), 2014 International Conference and Exposition on. IEEE*, 2014. p. 532-535.

[17] J. Tan, E-Health Care Information Systems, *Jossey-Bass Press*, 2005.

[18] https://tools.ietf.org/html/rfc7159 *Acessado em 06 de dezembro de 2018*.