Universidade de São Paulo–USP
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e de Computação
Curso de Engenharia Elétrica - Ênfase em Sistemas de Energia e Automação

Lucas Carlos Barboza

MODELO DE ARQUITETURA BASEADO EM UM SISTEMA DE INTERNET DAS COISAS APLICADA A AUTOMAÇÃO RESIDENCIAL

São Carlos
2015

Lucas Carlos Barboza

MODELO DE ARQUITETURA BASEADO EM UM SISTEMA DE INTERNET DAS COISAS APLICADA A AUTOMAÇÃO RESIDENCIAL

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

Curso de Engenharia Elétrica com ênfase em Sistemas de Energia e Automação

Orientador: Prof. Dr. Marcelo Andrade da Costa Vieira

São Carlos
2015

BARBOZA, LUCAS CARLOS
B238m MODELO DE ARQUITETURA BASEADO EM UM SISTEMA DE INTERNET DAS COISAS APLICADA A AUTOMAÇÃO RESIDENCIAL / LUCAS CARLOS BARBOZA; orientador Marcelo Andrade da Costa Vieira . São Carlos, .

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, .

1. Internet das Coisas. 2. Automação Residencial. 3. Modelagem de Sistemas. I. Título.

FOLHA DE APROVAÇÃO

Nome: Lucas Carlos Barboza

Título: "Automação residencial utilizando smartphone"

Trabalho de Conclusão de Curso defendido e aprovado em 25/06/2015,

com NOTA 7,5 (Sete, Cinco), **pela Comissão Julgadora:**

Prof. Dr. Marcelo Andrade da Costa Vieira - (Orientador - SEL/EESC/USP)

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

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

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Este trabalho é dedicado a toda minha família
por todo apoio e compreensão durante toda minha vida.

Agradecimentos

Agradeço primeiramente à minha família: a meu pai, Rinaldo, que sempre me incentivou, apoiou e confiou em todas as minhas decisões; à minha mãe, Débora, pelo seu exemplo de vida e superação que, por muitas vezes, foi onde busquei forças e inspiração para continuar; às minhas irmãs, Carolina e Gabriela, pelo seu amor, carinho e companheirismo; às minhas avós, Neusa e Jeanete, por me acolher quando precisei.

À Marina Vidual, minha companheira, pelo apoio e compreensão nos momentos mais difíceis, nos intermináveis finais de semana de estudo e trabalho.

Ao meu orientador Prof. Dr. Marcelo Andrade da Costa Vieira, pela confiança no trabalho e pelos conselhos durante a execução.

Ao meu primo Felipe Luiz Tortella e ao amigo Murilo Frônio Bássora, pela imensa ajuda nas áreas que mais tive dificuldade.

Por fim, a todos os que contribuíram, direta ou indiretamente, para a realização deste trabalho.

"Obstacles are those frightful things you see
when you take your eyes off your goal."
(Henry Ford)

Resumo

Barboza, Lucas **MODELO DE ARQUITETURA BASEADO EM UM SISTEMA DE INTERNET DAS COISAS APLICADA A AUTOMAÇÃO RESIDENCIAL**. 84 p. Trabalho de Conclusão de Curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, 2015.

O movimento onde objetos do dia a dia, antes inanimados, adquirem funcionalidades e conectam-se a internet é conhecido como Internet das Coisas. Esse trabalho tem como objetivo propor uma arquitetura de um sistema de internet das coisas aplicado à automação residencial. No futuro, casas inteligentes serão "conscientes" a respeito das coisas que acontecem dentro das mesmas, impactando, principalmente, em alguns aspectos como utilização inteligente de recursos, segurança e conforto. Através dessa arquitetura foi possível desenvolver um sistema configurável, sem fio e de fácil instalação de dispositivos distribuídos que adicionam funcionalidades de monitoramento e controle aos componentes presentes em uma casa. Como ponto de partida, foram realizadas algumas modelagens descritas pelo projeto IoT-A. A partir dessas modelagens foi possível chegar em um sistema dividido em quatro componentes, referenciados durante o trabalho como Central de Controle, Aplicação Web, Aplicação Móvel e Periféricos. Cada componente foi discutido individualmente, com a mentalidade nos requisitos propostos e também pensando em escalabilidade, baixo custo e facilidade de desenvolvimento. O resultado desse trabalho foi uma arquitetura de um sistema de internet das coisas aplicada a automação residencial, esses conceitos e o modelo foram validados através de um protótipo, instalado em uma maquete, que demonstram e comprovam a viabilidade da arquitetura do sistema desenvolvido.

Palavras-chave: Casa Inteligente. Automação Residencial. Internet das Coisas..

Abstract

Barboza, Lucas **ARCHITECTURE MODEL BASED ON AN INTERNET OF THINGS SYSTEM APPLIED TO HOME AUTOMATION**. 84 p. Final Paper – São Carlos School of Engineering, University of São Paulo, 2015.

The movement where objects of everyday life acquire functionality and connect to the internet is known as the Internet of Things. This work aims to propose an architecture of an internet of things system applied to home automation. In the future, smart homes will be "aware" about things that happen within them, mainly affecting in some respects as using intelligent features, security and comfort. Through this architecture was possible the development of a configurable, wireless and easy to install system of distributed devices that add monitoring and control functionality to the components present in a home. As a starting point, there were some modeling described by the IoT-A project. From these modeling was reached on a system divided into four components, referenced during this work as Control Center, Web Application, Mobile Application and Peripherals. Each component was discussed individually with the mentality on the proposed requirements and also thinking about scalability, low cost and ease of development. The result of this work was an architecture of an internet of things system applied to home automation, these concepts and the model were validated through a prototype installed on a model, which demonstrate and prove the feasibility of the developed system architecture.

Keywords: Smart Homes. Home Automation. Internet of Things.

Lista de ilustrações

Figura 1 Melhores Práticas - Colunas e Blocos 29
Figura 2 Modelo de Domínio 30
Figura 3 Modelo de Informação 32
Figura 4 Modelo MVC 37
Figura 5 Ciclo de vida de uma Activity 39

Figura 6 Arduino Uno 46
Figura 7 Dragino Yún Shield 47
Figura 8 Dragino Yún Shield Struct 48
Figura 9 Módulo nRF24L01+ 49
Figura 10 Modelo de Domínio do Sistema 51
Figura 11 Modelo de Informação do Controlador 52
Figura 12 Modelo de Informação do Periférico 53
Figura 13 Visão Geral 54
Figura 14 Modelo do Banco de Dados 56
Figura 15 Fluxograma da Central de Controle 60
Figura 16 Fluxograma do Periférico Interruptor Inteligente 63
Figura 17 Tela de login da Aplicação Web 66
Figura 18 Tela de cadastro da Aplicação Web 67
Figura 19 Tela de Centrais de Controle da Aplicação Web 67
Figura 20 Tela da Central de Controle da Aplicação Web 68
Figura 21 Tela de Periféricos da Aplicação Web 68
Figura 22 Tela de Novo Periféricos Aplicação Web 69
Figura 23 Tela de login da Aplicação Móvel 70
Figura 24 Tela de Centrais de Controle da Aplicação Móvel 70
Figura 25 Tela de Periféricos da Aplicação Móvel 71
Figura 26 Tela do Periférico da Aplicação Móvel 71
Figura 27 Maquete com o protótipo 72

Figura 28 Tabela de usuários 73
Figura 29 Tabela de Centrais de Controle 73
Figura 30 Tabela de Periféricos 74
Figura 31 Download dos dados da Central de Controle 74
Figura 32 Upload dos dados da Central de Controle 74
Figura 33 Requisição dos estados dos sensores da Aplicação Móvel 75
Figura 34 Controle de um Periférico pela Aplicação Móvel 75
Figura 35 Periférico Controlado pela Aplicação Móvel 76
Figura 36 Tela de um Periférico com leitura dos sensores 77
Figura 37 Tela de um Periférico com leitura dos sensores 77

Lista de tabelas

Tabela 1 Exemplo de Tabela de Dados 60

Lista de siglas

API *Application Programming Interface* — Interface de Programação de Aplicação

ARM *Architectural Reference Model*

ARPA *Advanced Research Projects Agency* — Ponto de Acesso

CE *Chip Enable*

CoC *Convention over Configuration*

CRC *Cyclic Redundancy Check*

CS *Chip Select*

DRY *Don't Repeat Yourself*

ESB *Enhanced ShockBurst*

GUI *Graphic User Interface* — Interface Gráfica de Usuário

GFSK *Gaussian Frequency-Shift Keying* — Terra

HTTP *Hypertext Transfer Protocol* — Protocolo de Transferência Hipertexto

ICSP *In Crcuit Serial Programming*

ID *Identification*

IDE *Integrated Development Environment* — Ambiente integrado de desenvolvimento

IoT *Internet of Things* — Internet das Coisas

ISM *Industrial, Scientific and Medical* — Industrial, Científica e Médica

IP *Internet Protocol*

JSON *JavaScript Object Notation*

LAN *Local area network* — Rede de área local

MVC *Model-view-controller* — Modelo-visão-controlador

MVP *Minimum Viable Product* — Produto Mínimo Viável

MISO *Master In Slave Out*

MOSI *Master Out Slave In*

PWM *Pulse-Width Modulation* — Modulação em Largura de Pulso

PaaS *platform as a Service*

RAM *Random Access Memory*

REST *Representational State Transfer* — Transferência de Estado Representacional

RF *Radio Frequency* — Computador com um conjunto reduzido de instruções

SCK *Serial Clock*

SDK *Software Development Kit* — Kit de Desenvolvimento de Software

SoC *System-on-a-chip*

SPI *Serial Peripheral Interface* — Linguagem de Consulta Estruturada

SS *Slave Select*

SSH *Secure Shell*

TCP *Transmission Control Protocol* — Lógica Transistor-Transistor

UI *User Interface* — Interface de Usuário

URI *Uniform Resource Identifier* — Identificador Uniforme de Recursos

URL *Uniform Resource Locator*

URN *Uniform Resource Name*

USB *Universal Serial Bus*

ULP *Ultra Low Power*

XML *eXtensible Markup Language*

Sumário

1 Introdução **23**

1.1 Objetivos 24

1.2 Justificativa 24

2 Fundamentação Teórica **27**

2.1 Projeto IoT-A 27

2.1.1 Modelo de Arquitetura de Referência 28

2.2 Internet 33

2.2.1 TCP/IP 34

2.2.2 HTTP 35

2.3 Comunicação sem fio 36

2.4 MVC 37

2.5 API 38

2.6 Android 38

3 Materiais e Métodos **41**

3.1 Materiais 41

3.1.1 Ruby 41

3.1.2 Ruby-on-Rails 42

3.1.3 Biblioteca SPI 43

3.1.4 Biblioteca RF24 44

3.1.5 Biblioteca Bridge 45

3.1.6 Hardware 45

3.2 Métodos 50

3.2.1 Modelagem do Sistema 50

3.2.2 Componentes Do Sistema 53

3.2.3 Aplicação Web 55

3.2.4 Central de Controle 59

3.2.5 Periféricos 62
3.2.6 Aplicação Móvel 64
3.2.7 Relacionamento dos componentes com o Modelo de Domínio . . . 66

4 Resultados **73**

5 Discussões e Conclusão **79**
5.1 Discussões 79
5.1.1 Aplicação Web 79
5.1.2 Central de Controle 80
5.1.3 Periféricos 81
5.1.4 Aplicação Móvel 81
5.2 Conclusão 81

Referências Bibliográficas **83**

Capítulo 1

Introdução

A internet das Coisas é o movimento onde objetos do cotidiano passam a se conectar e trocar informações através da internet. São objetos físicos, "coisas", que passam a alojar sistemas eletrônicos embarcados com componentes de *software*, sensores e, principalmente, conectividade, que permite esses objetos trocarem informações através da rede. Esses tecnologias permitem que os objetos possam ser controlados e monitorados remotamente e possibilitam uma maior integração entre sistemas baseados em computadores, que também estão na rede, e o mundo físico.

A automação residencial, também conhecida como domótica, é a integração entre vários equipamentos motorizados e automatizados capazes de trocar informações entre si. O setor, que cresce a cada dia, vem com a finalidade de facilitar a vida das pessoas, satisfazendo as suas necessidades de comunicação, conforto e segurança. Computadores de uso geral não são mais a maioria dos dispositivos que conectam-se à internet, isso pode ser observado na rotina das pessoas, que estão cada vez mais acostumadas com objetos conectados, e a interagirem com esses objetos através de dispositivos móveis e interfaces web que permitem controlar o ambiente que os cercam, através de interfaces cada vez mais amigáveis e interativas.

Essa tecnologia pode proporcionar aos seus consumidores um conforto antes não imaginado e pode ser facilmente adaptada a qualquer utilidade doméstica. Além da comodidade de poder controlar e monitorar remotamente vários serviços, é possível diminuir o tempo gasto com tarefas rotineiras, otimizando o tempo do usuário, como, por exemplo, apagar as luzes em horários pré-definidos, ligar sistemas de irrigação quando necessário, trancar as portas, otimizar o controle de ar-condicionado, evitar desperdícios desligando sistemas automaticamente quando não são necessários. Em suma, os sistemas embarcados possibilitam transferir as decisões referentes a essas tarefas rotineiras aos microcontroladores, que também serão responsáveis pela leitura, através de sensores, do ambiente e interpretação desses dados para que possam, assim, atuar sobre o mesmo. As aplicações são inumeráveis, cabe ao usuário definir suas necessidades.

1.1 Objetivos

O objetivo do presente trabalho é a definição da arquitetura de um sistema de *Internet of Things* (IoT) para uma aplicação em automação residencial. Os conceitos presentes no âmbito da IoT de escalabilidade e modularidade foram aplicados ao sistema desenvolvido. A arquitetura desenvolvida deve permitir colocar a disposição do usuário a decisão do tamanho do projeto de automação que será instalado em sua residência, sendo necessário escolher quais e quantos Periféricos deseja instalar, tarefa essa que será realizada pelo próprio consumidor, também será possível acrescentar ou retirar Periféricos conforme seja constatada uma necessidade maior ou menor do projeto inicial.

A partir do modelo produzido, outro objetivo foi a criação um protótipo, que foi utilizado para validar os conceitos e a estrutura da arquitetura do sistema. Para atingir esses objetivos, foram realizados estudos e modelagens do sistema como proposto pelo projeto IoT-A, que visa estabelecer as bases arquitetônicas de uma futura Internet das Coisas, e a partir destes modelos, escolhas de projeto adequadas foram tomadas para desenvolvimento do protótipo que valida o modelo. Para o desenvolvimento do protótipo, foram utilizadas plataformas de prototipagem *open-source*, tanto em *hardware* quanto em *software*, reduzindo assim os custos de desenvolvimento, uma vez que acelera o processo de implementação devido aos esforços da comunidade *open-source* em partilhar projetos e conhecimento.

1.2 Justificativa

"A Internet das Coisas tem potencial para criar um impacto econômico de 2.7 trilhões a 6.2 trilhões de dólares anualmente até 2025." (MANYIKA et al., 2013).

> A Internet das Coisas é a próxima revolução na computação. Enquanto smartphones e a internet móvel viu o advento de aplicativos para o consumidor, esperamos ver a integração generalizada de semicondutores, comunicação móvel e Big Data impulsionando a Internet das Coisas na economia em geral. (JUNGLING; WOOD, 2014)

"Ninguém assumiu o controle dessas áreas aqui no Brasil. Vale a pena considerar essas novas tecnologias como forma de disruptar mercados e criar novas ideias para tirar ineficiências e atender a carência de serviços de qualidade que o Brasil tem."(RIVA, 2014).

É fato que esse mercado tem muito a ser explorado, esse projeto se justifica por gerar conhecimento e experiência em uma área que ainda crescerá muito nos próximos anos (BRADLEY; BARBIER; HANDLER, 2013) e também resultou em um protótipo de um possível produto competitivo no mercado atual Brasileiro, na área de Internet das Coisas e, mais especificamente, na área de automação residencial.

Um dos fatores que influenciam a decisão das pessoas a não aderirem a essas tecnologias, instalarem algum tipo de sistema automatizado em suas residências e explorarem

esse mundo totalmente novo, é a necessidade de instalações complexas, sendo por vezes necessário quebrar paredes, novos cabeamentos e possíveis alterações na estrutura da casa. Esse tipo de serviço resulta em alto custo de instalação e limita o acesso à essas tecnologias a entusiastas que tenham boas condições financeiras. Esse projeto então se justifica por desenvolver um sistema de automação residencial de baixo custo e fácil instalação.

Capítulo 2

Fundamentação Teórica

2.1 Projeto IoT-A

Essa seção propõe um estudo sobre o projeto IoT-A (BAUER et al., 2011), explicando os principais objetivos desse projeto e, após a introdução dos objetivos, conceitos básicos e como utilizar esse documento, são exploradas com maior nível de detalhamento duas modelagens que utilizadas para desenvolver o modelo proposto nesse trabalho.

Hoje, o *slogan* "Internet das Coisas" é utilizado por muitos produtos. Essas soluções oferecem pouca ou nenhuma capacidade de interagir com outros produtos, uma vez que são desenvolvidos para solucionar somente um problema específico. Além disso, o cenário da IoT cobre aplicações em campos completamente diferentes, sendo assim, as tecnologias utilizadas variam muito de uma aplicação para a outra. Portanto, as soluções são verticais e podem ser chamadas de "INTRAnet das Coisas" ao invés de "INTERnet das Coisas", no longo prazo esse modelo de desenvolvimento é insustentável. Assim como os protocolos no campo das redes, onde surgiram vários diferentes no início, todos eles deram lugar a um modelo comum, chamado de TCP/IP. O surgimento de um modelo de referência para o cenário IoT e a identificação de arquiteturas de referência pode conduzir a um desenvolvimento mais ágil e focado possibilitando assim um crescimento exponencial das soluções relacionadas à IoT (BAUER et al., 2011). Levando em consideração apenas os aspectos técnicos, as soluções IoT dos dias de hoje não abordam os requerimentos de escalabilidade de uma futura IoT, tanto em questões de comunicação entre dispositivos diferentes e capacidade de gerenciar os mesmos.

Essas considerações levaram à criação de, primeiramente, um Modelo de Referência para o domínio IoT com o objetivo de fornecer um entendimento comum sobre os problemas relacionados à esse domínio. Em segundo lugar, fornecer para as empresas que desejam criar suas próprias soluções IoT um suporte através de uma Arquitetura de Referência que descreve os blocos essenciais que essa solução deve conter, assim como guia as escolhas de *design* ideais para lidar com os requisitos, que conflitam entre si, de desempenho, funcionalidades, implementação e segurança. (BAUER et al., 2011)

Não é escopo desse trabalho de conclusão de curso seguir toda a metodologia proposta pelo projeto IoT-A, mas sim um estudo inicial sobre a utilização do documento e, a partir disso, a criação de dois modelos que compõe o início do processo, tratados nas seções subsequentes.

2.1.1 Modelo de Arquitetura de Referência

Primeiramente, faz-se necessário uma análise da visão base do projeto IoT-A, mostrando como estes modelos são utilizados para que uma arquitetura seja definida. É exibida também a Modelagem de Domínio, a qual não detalha, mas mostra todos os tópicos relevantes que devem ser analisados no desenvolvimento do projeto de um sistema voltado à Internet das Coisas.

O IoT-A descreve blocos essenciais e escolhas de *design* para lidar com os requerimentos, que conflitam entre si, de funcionalidades, desempenho, agilidade no desenvolvimento e segurança. Uma Arquitetura de Referência pode ser vista como uma "Matriz" que, idealmente, é a base para todas as arquiteturas concretas. Para estabelecer tal Matriz, com base numa análise forte e exaustiva do Estado da Arte, foi necessário analisar o conjunto de todas as possíveis funcionalidades, mecanismos e protocolos que podem ser utilizados para construir essas arquiteturas concretas. Através dessa base de conhecimento, juntamente com um conjunto de possíveis escolhas de *design*, baseado em uma caracterização do sistema a ser desenvolvido, é possível aos arquitetos de sistemas escolherem entre os protocolos, componentes funcionais e opções de arquitetura necessários para criar seus sistemas voltados para a IoT.

A definição de uma arquitetura concreta de um sistema, de acordo com o IoT-A, é realizada através de uma série de melhores práticas que guiam o arquiteto do sistema através das modelagens. O objetivo principal é derivar uma arquitetura para um domínio específico a partir do *Architectural Reference Model* (ARM). Essas melhores práticas podem ser divididas em três grandes blocos (Figura 1), pilares principais. São eles:

- Manual: Constituída por sete blocos onde, os quatro primeiros, correspondem à utilização dos modelos propostos pelo projeto IoT-A: (1) Modelo de Domínio. (2) Modelo da Informação. (3) Modelo Funcional. (4) Modelo de Comunicação. Além desses, existem os blocos referentes à perspectivas, desenvolvimento e segurança.

- Passo: Composto por três blocos, um sobre as escolhas de *design* que podem ser utilizadas nas modelagens do item anterior, outro sobre os requerimentos de desenvolvimento do sistema e uma análise dos riscos envolvendo o projeto.

- Ilustração: O terceiro, e último, pilar consiste de dois blocos, um que busca investigar a generalidade do modelo IoT proposto e outro sobre Casos de Uso que analisa as aplicações específicas do domínio proposto.

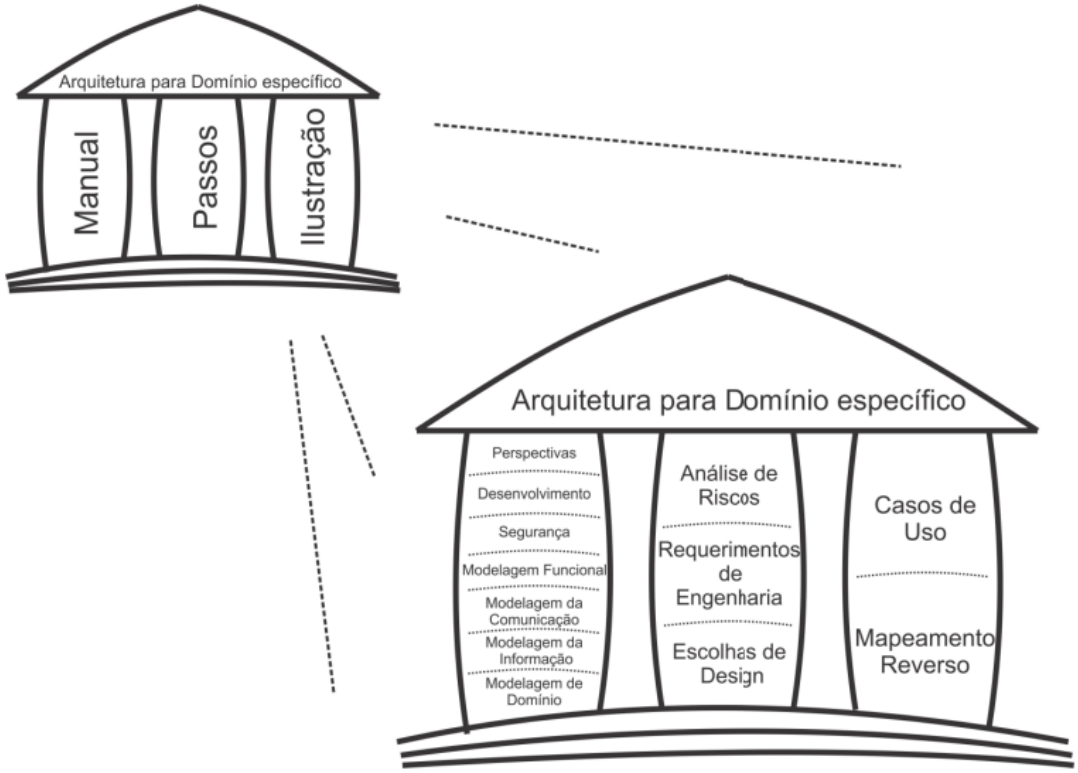

Figura 1: Melhores Práticas - Colunas e Blocos.
Fonte: Adaptado de MAGERKURTH, 2013.

Nas próximas seções serão explicados os dois modelos que foram usados para a criação do projeto proposto nesse trabalho.

2.1.1.1 Modelo de Domínio

O Modelo de Domínio, representado pela Figura 2, proposto pelo projeto IoT-A, define as descrições dos conceitos pertencentes a uma área de interesse em particular, também é no mesmo modelo onde os atributos básicos desses conceitos como, por exemplo, nome e identificador são definidos. Outro importante aspecto definido desse modelo são as relações entre os conceitos definidos.

O Modelo de Domínio proposto é uma parte importante de qualquer modelo de referência, pois o mesmo inclui a definição de vários conceitos abstratos, suas responsabilidades e suas relações. Ele não trata de particularidades das tecnologias que podem vir a ser utilizadas, mas sim as relações entre as entidades físicas e suas representações virtuais, que, através da troca de informações, disponibilizarão serviços e recursos aos usuários.

O principal objetivo de um modelo de domínio é gerar um entendimento comum do sistema alvo em questão. Tal entendimento comum é importante, não apenas internamente ao projeto, mas também para o discurso científico. Somente com um entendimento comum dos conceitos principais, descritos no modelo de domínio, é possível avaliar e discutir soluções arquitetônicas (BAUER et al., 2013). Esse modelo, então, representa de forma

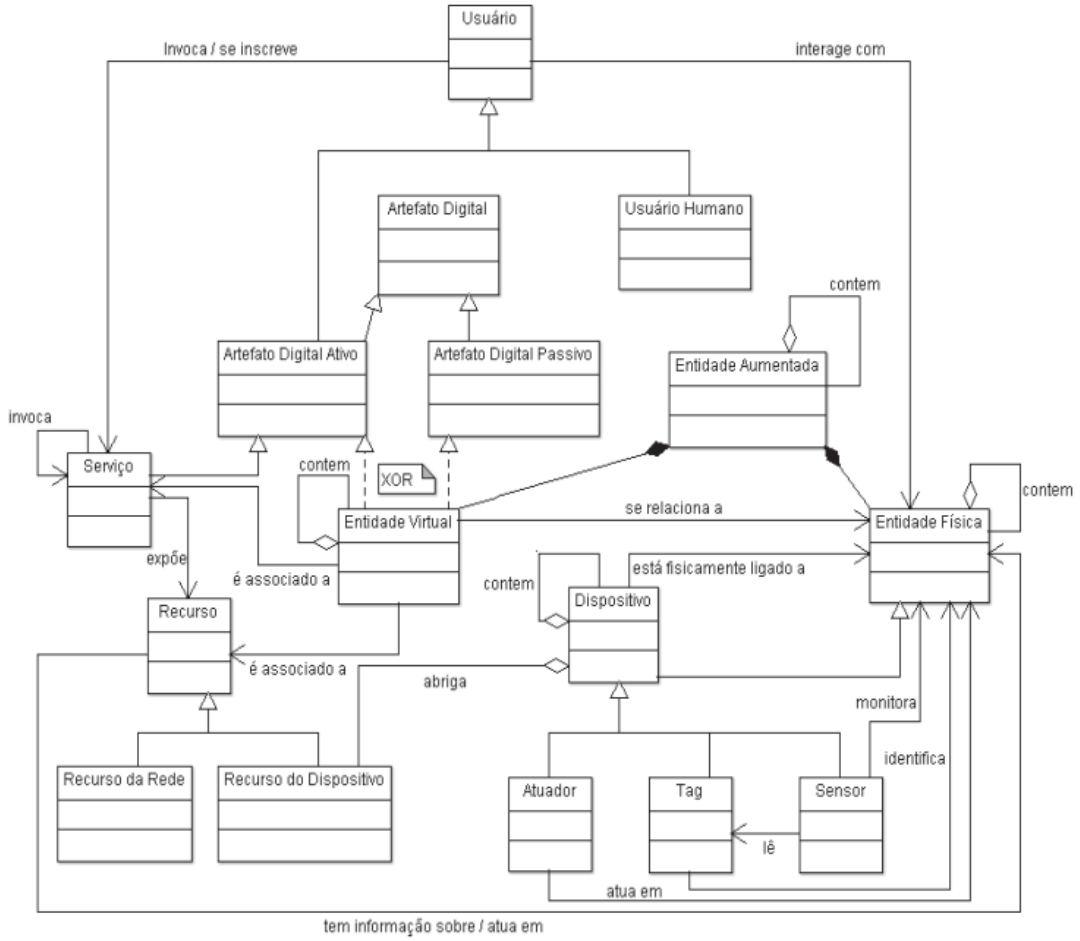

Figura 2: Modelo de Domínio.
Fonte: Adaptado de MAGERKURTH, 2013.

geral, como um cenário da IoT deve ser modelado para que todos os aspectos técnicos sejam considerados para um domínio específico. Essa modelagem é o primeiro passo para criar um Modelo de Referência.

Uma vez introduzido o modelo, é importante explicar os conceitos envolvidos na modelagem do mesmo. Um cenário para a IoT pode ser descrito, de forma genérica, como um usuário que interage com Entidades Físicas, "coisas" no mundo real. Pode-se definir o usuário de um sistema IoT como um humano utilizando o sistema ou um Artefato Digital. Quando o usuário é um ser humano, este acessará os serviços através de interfaces em softwares disponíveis para o mesmo. Entidades Físicas podem ser qualquer "coisa" do mundo real que precise ser representada no modelo.

Entidades Físicas são representadas no mundo virtual através de Entidades Virtuais. Existem muitos tipos de representações digitais das entidades físicas, dentre elas é possível listar:

- Modelos 3D.
- Avatares.

- ❏ Entradas em Banco de Dados.
- ❏ Objetos ou instâncias de uma classe em uma linguagem de programação orientada a objetos.

As Entidades Virtuais possuem duas propriedades fundamentais:

- ❏ Artefatos Digitais: Entidades Virtuais representam apenas uma entidade física, mas entidades físicas podem estar a elas associadas inúmeras entidades virtuais. Cada entidade virtual possui uma *Identification* (ID). É possível visualizar as Entidades Virtuais como artefatos digitais, que podem ser passivos, como, por exemplo, uma entrada em um banco de dados ou ativos, componentes de *software* que acessam outros serviços ou recursos do sistema.
- ❏ Idealmente, Entidades Virtuais são representações sincronizadas de uma série de aspectos (ou propriedades) da Entidade Física, isso significa que parâmetros digitais relevantes, que representam as características da entidade física, são atualizados na medida que mudanças ocorrem.

Uma Entidade Aumentada é a composição de uma Entidade Virtual e uma Entidade Física a qual ela está associada. Essa definição é feita para destacar o fato de que esses conceitos estão intimamente relacionados, são essas entidades que permitem que objetos do nosso dia-a-dia possam interagir no mundo virtual, sendo parte do processo, constituindo as "coisas" do IoT.

A relação entre Entidade Física e Entidade Virtual é conseguida, normalmente, embarcando um ou mais dispositivos com capacidade de processamento computacional em objetos físicos comuns, permitindo assim controlá-los ou somente obter informações sobre o mesmo. Surge então o termo Dispositivo, nome dado ao conjunto, aumentando a capacidade de objetos comuns, permitindo que os mesmos realizem tarefas mais inteligentes e interajam com o mundo virtual.

Do ponto de vista da IoT existem três tipos básicos de dispositivos, são eles:

- ❏ Sensores: coletam informações sobre as Entidades Físicas que monitoram.
- ❏ Tags: atribuem uma identificação única, perante ao sistema, as Entidades Físicas às quais estão associadas.
- ❏ Atuadores: dispositivo capaz de alterar o estado da Entidade Física.

Recursos são componentes de software que providenciam informações ou são usados na ativação de atuadores nas Entidades Físicas. Existem, basicamente, dois tipos de recursos, são eles: (1) Recursos do Dispositivo, quando esses componentes de *software* estão hospedados no próprio Dispositivo associado à Entidade Física. (2) Recursos de Rede,

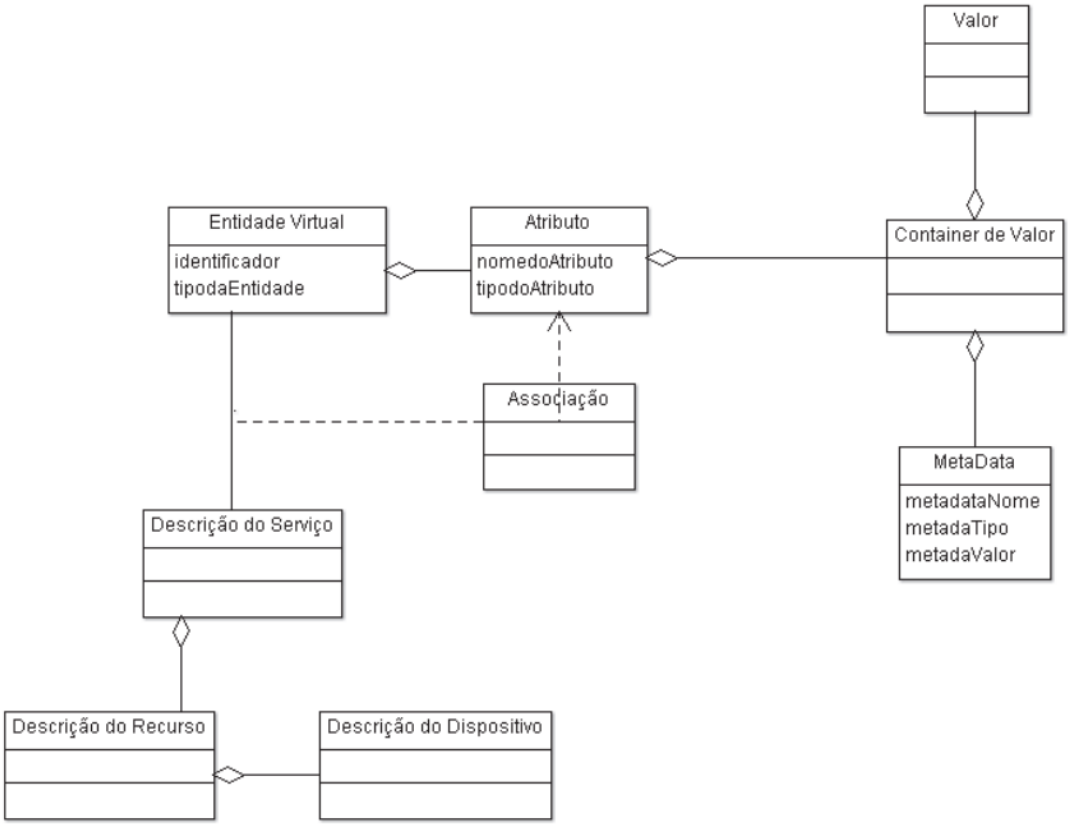

Figura 3: Modelo de Informação.
Fonte: adaptado de MAGERKURTH, 2013.

quando os recursos estão disponíveis através da rede, internet ou intranet, armazenados em algum servidor web.

Os recursos são oferecidos através de serviços, que disponibilizam uma interface bem definida e padronizada oferecendo todas as funcionalidades necessárias para a interação com Entidades Físicas. A grosso modo, serviços são responsáveis por expor as funcionalidades, recursos, de um Dispositivo.

Esses conceitos serão utilizados no Capítulo 3, onde um Modelo de Domínio para o sistema proposto é criado. Na próxima seção, outra modelagem do IoT-A será explicada, o Modelo de Informação.

2.1.1.2 Modelo de Informação

Outro aspecto tratado pela modelagem proposta pelo projeto IoT-A é a da informação. O Modelo de Informação, exemplificado na Figura 3, define a estrutura em que a informação é manipulada pelo sistema em um alto nível de abstração. A descrição da representação da informação (como por exemplo, descrição de XMLs ou RDFs e sua estrutura) não são parte deste modelo, mas sim um desdobramento importante.

O Modelo de Informação, proposto pelo IoT-A, detalha a modelagem de uma Entidade Virtual. Esse modelo tem expressiva relação com o Modelo de Domínio. Toda Entidade

Virtual possui um tipo, nome e um ou mais valores aos quais meta-informações podem estar associadas.

As Entidades Virtuais, Descrições dos Serviços e Associações são os principais componentes tratados por esse modelo. Como explicado na seção anterior, sobre o Modelo de Domínio, as Entidades Virtuais estão associadas às Entidades Físicas e a Descrição de um Serviço, como o nome indica, descreve um serviço que está associado àquela Entidade Física. Associações servem para modelar a conexão entre um atributo de uma Entidade Virtual e a Descrição de um Serviço.

Cada Entidade Virtual possui, perante ao sistema, um identificador único e um Tipo de Entidade, que pode ser um ser humano, sensor, atuador e etc. O Tipo de Entidade pode ter relação com conceitos que definem que tipo de atributo uma Entidade Virtual deve ter. Cada Contêiner de Valor pode conter de 0 a n metadados associados.

Uma Descrição de Serviço descreve as características relevantes de um Serviço, incluindo a sua interface. Descrição de Recursos descrevem um recurso, o qual possui suas funcionalidades descritas por um serviço, a Descrição de Recurso pode conter informações sobre o Dispositivo no qual o recurso está hospedado, caso o recurso seja *On-Device*.

2.2 Internet

Uma *network*, em português rede, é um grupo de dispositivos conectados e que são capazes de conversar um com o outro. Uma *internet* são duas ou mais redes que conseguem se comunicar entre si. A Internet, é a mais notável das *internets*, composta por centenas de milhares de redes conectadas. Indivíduos particulares, organizações governamentais, escolas, universidades, empresas privadas e públicas, bibliotecas em praticamente todos os países utilizam a Internet. Milhões de pessoas são usuários, apesar de esse sistema ter iniciado em 1969. (FOROUZAN, 2010)

Em meados de 1960, os computadores *mainframes* utilizados em pesquisas eram dispositivos *stand-alone*, ou seja, que operavam sozinhos. Computadores de fabricantes diferentes não eram capazes de comunicar entre si. A *Advanced Research Projects Agency* (ARPA) do Departamento de Defesa dos EUA, tinha interesse em encontrar uma forma de criar uma comunicação entre dois computadores para que as pesquisas pudessem ser compartilhadas , a fim de evitar esforços em dobro dos pesquisadores e reduzir assim os custos. Então, em 1967 a ARPA apresentou suas ideias para a ARPANET, uma pequena rede de computadores conectados. Em 1969 a ARPANET era uma realidade, com quatro pontos de acesso: a Universidade da Califórnia em Los Angeles, a Universidade da Califórnia em Santa Bárbara, o Instituto de Pesquisas de Stanford e a Universidade de Utah estavam conectados. (FOROUZAN, 2010)

A partir desse ponto, outras redes começaram a ser criadas e ficou cada vez mais evidente a necessidade em criar padrões para que essas redes pudessem se comunicar umas

com as outras. Em 1973 começaram então os primeiros esforços parar criar esse protocolo, um artigo chamado *Transmission Control Protocol* (TCP) surgiu como estudos para resolver os problemas de comunicação entre diferentes fabricantes. Em 1977 ficou demonstrado o sucesso desse protocolo criando uma *internet* com três redes, ARPANET, PacketRadio e PacketSatellite. O Protocolo TCP foi então dividido em dois, TCP e *Internet Protocol* (IP) e ficou conhecido como TCP/IP, que é explicado mais detalhadamente a seguir.

2.2.1 TCP/IP

O TCP/IP é um protocolo com cinco camadas, sendo quatro delas camadas de *software* e uma de *hardware*, as camadas TCP/IP foram nomeadas similarmente ao Modelo OSI. A diferença entre os dois reside no fato de que no TCP/IP as camadas seis e sete do Modelo OSI foram incorporadas juntamente à camada 5, chamada de Aplicação (FOROUZAN, 2010).

A Camada Física do protocolo TCP/IP não define nenhum protocolo específico, mas suporta todos os padrões e protocolos proprietários. Nesse ponto, a comunicação é puramente entre dois nós consecutivos, que podem ser computadores ou roteadores. Aqui a unidade da comunicação é um único bit, transportado através da conexão entre os dois dispositivos, essa camada trata os bits individualmente.

A segunda camada, chamada de Enlace do protocolo TCP/IP também não define nenhum protocolo específico, mas suporta todos os padrões e protocolos proprietários. Nesse nível a comunicação ainda é entre dois nós específicos, consecutivos, a diferença é que aqui, a unidade de comunicação é um *Frame*, um pacote que encapsula os bits recebidos utilizando a camada física, em dados com um cabeçalho. O cabeçalho, além de outras informações a respeito dos dados transmitidos, contém as informações de quem é o destinatário e a fonte do *Frame*.

A Camada de Rede, a terceira do protocolo TCP/IP suporta, além de outros, o protocolo IP. Neste nível a comunicação não é mais entre dois nós consecutivos na rede, mas sim entre dois pontos, nas duas primeiras camadas a comunicação é fim-a-fim e, na camada de rede, a comunicação é ponto-a-ponto. A unidade de comunicação é um Datagrama, que trata-se de uma entidade completa de dados, independente e que contém informações suficientes para ser roteada da origem ao destino sem precisar de informações adicionais.

A próxima camada, chamada Camada de Transporte é a quarta do protocolo TCP/IP. Nessa camada é definido o protocolo TCP, um protocolo que realiza, além da multiplexação dos dados, uma série de funções para tornar a comunicação entre dois pontos mais confiável. Até a camada anterior, os dados não eram ordenados e, os Datagramas por si só não fazem sentido, a camada de transporte faz a ordenação dos pacotes recebidos, garante que não houve perda de pacotes e os dados são íntegros. A grande diferença é que na

Camada de Rede, a comunicação é feita ponto-a-ponto, mas de um Datagrama, enquanto na Camada de Transporte, a comunicação ponto-a-ponto é de mensagens inteiras.

Por último, a Camada de Aplicação, quinta do protocolo TCP/IP. Esta camada permite ao usuário acessar os serviços da rede. São muitos os protocolos definidos na Camada de Aplicação. Na próxima seção será detalhado um protocolo dessa camada de interesse para o desenvolvimento desse projeto, o HTTP.

2.2.2 HTTP

O *Hypertext Transfer Protocol* (HTTP) é um protocolo de comunicação, presente na Camada de Aplicação do Modelo OSI, com a leveza e velocidade necessária para sistemas de comunicação hipermídia distribuídos e colaborativos. É um protocolo genérico, *stateless* e orientado a objetos que pode ser usado em inúmeras aplicações, como por exemplo em servidores e sistemas de gerenciamento de objetos, através dos seus métodos de requisições (comandos). Uma das características do protocolo HTTP é o tipo da representação dos dados, que permite que sistemas sejam construídos independentemente dos dados que estão sendo transferidos. Vem sendo usado na *World-Wide Web* desde 1990. (BERNERS-LEE; FIELDING; FRYSTYK, 1996)

Na prática, sistemas de informação precisam de mais funcionalidades além das requisições simples, entre elas é possível citar pesquisas, atualização do *front-end* e comentários. O HTTP oferece vários métodos que possuem a finalidade de indicar o propósito de uma requisição, ele se baseia na referência da *Uniform Resource Identifier* (URI), como a localização *Uniform Resource Locator* (URL) ou o nome *Uniform Resource Name* (URN) para indicar em qual recurso um método será aplicado. (BERNERS-LEE; FIELDING; FRYSTYK, 1996)

Abaixo segue os termos que são utilizados para se referir às funções desempenhados pelos participantes ou objetos em uma comunicação HTTP:

- *Connection*: Uma representação virtual de uma conexão entre duas aplicações que se comunicam.
- *Message*: Unidade básica em uma comunicação HTTP, consiste de uma sequência de octetos estruturada que corresponde a uma sintaxe definida na documentação RFC1945 (BERNERS-LEE, 1996).
- *Request*: Uma mensagem HTTP de requisição.
- *Response*: Uma mensagem HTTP de resposta.
- *Resource*: Um objeto da rede ou serviço que pode ser identificado por uma URI.
- *Entity*: Representação particular ou interpretação dos dados, resposta ou recurso de um serviço, que pode ser encapsulado em uma mensagem de requisição ou resposta.

Consiste da meta-informação na forma de cabeçalhos ou conteúdo na forma de um corpo da mensagem.

- *Client*: Aplicação que estabelece uma conexão a fim de enviar requisições.
- *User Agent*: Um *client* que iniciou uma requisição.
- *Server*: Aplicação que aceita conexões a fim de responder as requisições HTTP.
- *Proxy*: Programa intermediário que age como um *client* e um *server* com o objetivo de fazer requisições em nome de outro *client*. Os *proxys* precisam interpretar as mensagens e, se necessário, reescrevê-las antes de passarem a diante. Normalmente são utilizados do lado do *client* como *network firewalls*.
- *Gateway*: Um *server* que age como um *client* intermediário para outro *server*.

O protocolo HTTP é baseado em um sistema de requisição e respostas. O cliente estabelece uma conexão com o servidor e envia uma solicitação na forma de um método de requisição, URI, e a versão do HTTP utilizada, seguido de uma mensagem MIME-*like*, extensões multi-função para mensagens de internet, contendo modificadores da requisição, informações sobre o cliente e possivelmente um conteúdo no corpo da mensagem. O servidor então responde com uma linha de *status* da requisição, incluindo a versão do protocolo utilizada e se foi recebida com sucesso ou qual erro encontrado ao processar a requisição, seguido de uma mensagem MIME-*like* contendo informações do servidor, meta-informação da entidade e possivelmente um conteúdo no corpo da mensagem. A grande maioria das comunicações HTTP são requisições de clientes para o servidor de origem, que contém os recursos ou serviços que o cliente busca.(BERNERS-LEE; FIELDING; FRYSTYK, 1996)

2.3 Comunicação sem fio

As comunicações *wireless* enviam sinais, pacotes de dados através do ar utilizando-se das ondas eletromagnéticas, na frequência das ondas de rádio, que variam de 3kHz até 300GHz. No caso do módulo nRF24L01 utilizado na comunicação entre a Central de Controle e os Periféricos, a frequência utilizada é a de 2.4GHz. A faixa de frequência utilizada está dentro das regiões que são mantidas abertas para uso genérico, chamada de banda *Industrial, Scientific and Medical* (ISM), é também a faixa de frequência utilizada por roteadores comuns residenciais, telefones sem-fio, microondas e etc. (BUTLER et al., 2013) Nesta seção serão apresentados alguns conceitos básicos sobre comunicação *wireless*.

Para que uma comunicação entre dispositivos *wireless* aconteça, é preciso que as informações estejam na forma de sinais elétricos para serem transmitidos por ondas eletromagnéticas. Esses sinais com as informações a serem transmitidas modulam uma onda

de rádio portador. Os sinais elétricos são variações de tensão elétrica, que, no caso desse projeto, são sinais digitais (MEDEIROS, 2007).

Uma comunicação sem fio precisa de dois adaptadores, um que transforma os dados a serem enviados em sinais elétricos e, posteriormente, ondas eletromagnéticas para então transmiti-los através de uma antena, e outro, da mesma natureza, capaz de fazer o caminho inverso.

2.4 MVC

MVC é o acrônimo de *Model-view-controller*, uma representação pode ser vista na Figura 4. Modelo-visão-controlador é um modelo de arquitetura de *software* que separa a representação da informação da interação do usuário ou aplicações com o sistema. A arquitetura *MVC* foi criada entre 1978 e 1979 por Trygve Reenskaug como uma solução para o problema geral de usuários controlando conjuntos de dados grandes e complexos, primeiramente para ser utilizada na *SmallTalk*, uma linguagem de programação orientada a objetos criada na Xerox, e depois evoluiu devido ao grande interesse pela estrutura criada.

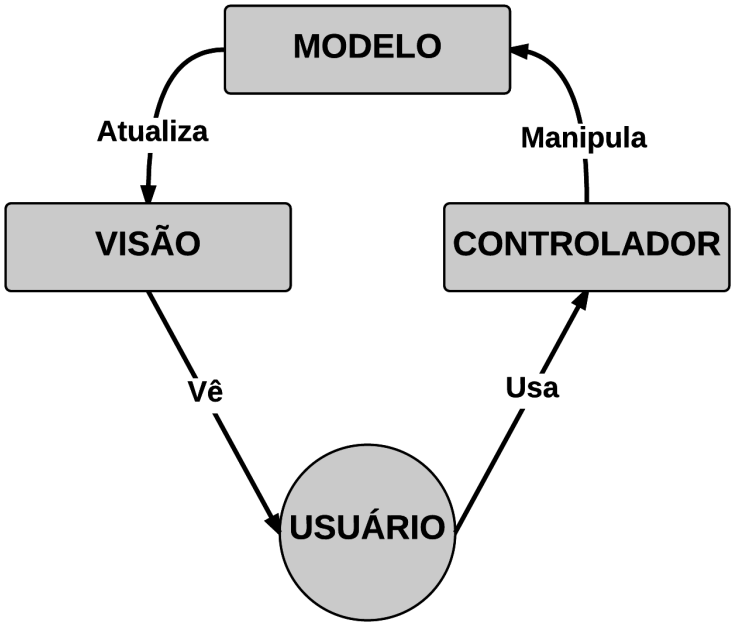

Figura 4: Modelo MVC.

O Modelo consiste de uma abstração da representação na forma de dados em um sistema computacional. Os Modelos são representados nos computadores como uma coleção

de dados junto aos métodos necessários para processar esses dados, cada Modelo está associado a um ou mais *Views*, em português, Visão.

A Visão é uma estrutura capaz de mostrar uma ou mais representações do Modelo na tela, ela pode mostrar uma ou mais características do Modelo que representa e/ou oculta outras, que pode ser o *front-ent* do usuário ou as aplicações utilizando as *Application Programming Interface* (API). A Visão também é capaz de realizar operações sobre os Modelos as quais estão associadas.

O Controlador é a ligação entre o usuário e/ou a aplicação e o sistema, ele faz a mediação da entrada, dos dados de entrada, convertendo-os em comandos para o Modelo ou dados que serão exibidos na Visão. Os *frameworks web* MVC colocam o modelo, a visão e o controlador totalmente no lado do servidor, os componentes *front-end* do sistema recebem somente os dados, e não a lógica por trás. (REENSKAUG, 1979)

2.5 API

Uma API é o acrônimo para *Application Programming Interface*, em português, Interface de Programação de Aplicativos e trata-se do conjunto de funções, rotinas e padrões estabelecidos por um programador no desenvolvimento de um *software* para que a utilização das suas funcionalidades e serviços por aplicativos externos, ou mesmo outros níveis da mesma aplicação, seja de fácil implementação e que não precisem envolver-se em detalhes da implementação do *software*, mas sim apenas utilizar os seus serviços e funcionalidades. (FOLDOC, 1995)

No ambiente *web*, uma API faz referência a um conjunto definido de mensagens e resposta de requisições HTTP, as quais podem ser expressas utilizando os formatos *eXtensible Markup Language* (XML) ou *JavaScript Object Notation* (JSON). Usualmente utilizando uma técnica chamada *Representational State Transfer* (REST), que pode ser resumida, nesse contexto, como uma sintaxe específica utilizada para construir as URI, cada recurso disponível pela API é unicamente direcionado através da sua URI.

2.6 Android

O Android é um Sistema Operacional que possui seu núcleo baseado no *kernel* Linux, esse sistema, desenvolvido pelo Google, tem seu foco em dispositivos móveis com telas sensíveis ao toque, como é o caso de *smartphones* e *tablets*, apesar de possuir outras distribuições específicas para televisões, carros, relógios e etc.

Uma das principais classes no desenvolvimento de um aplicativo Android são as *Activitys*. Uma *activity* é, basicamente, uma classe que trata do gerenciamento de uma *User Interface* (UI). Todo aplicativo Android inicia-se através de uma *activity*, ou seja, quando um aplicativo é iniciado, sua *Activity* principal é chamada. Um aplicativo pode contar

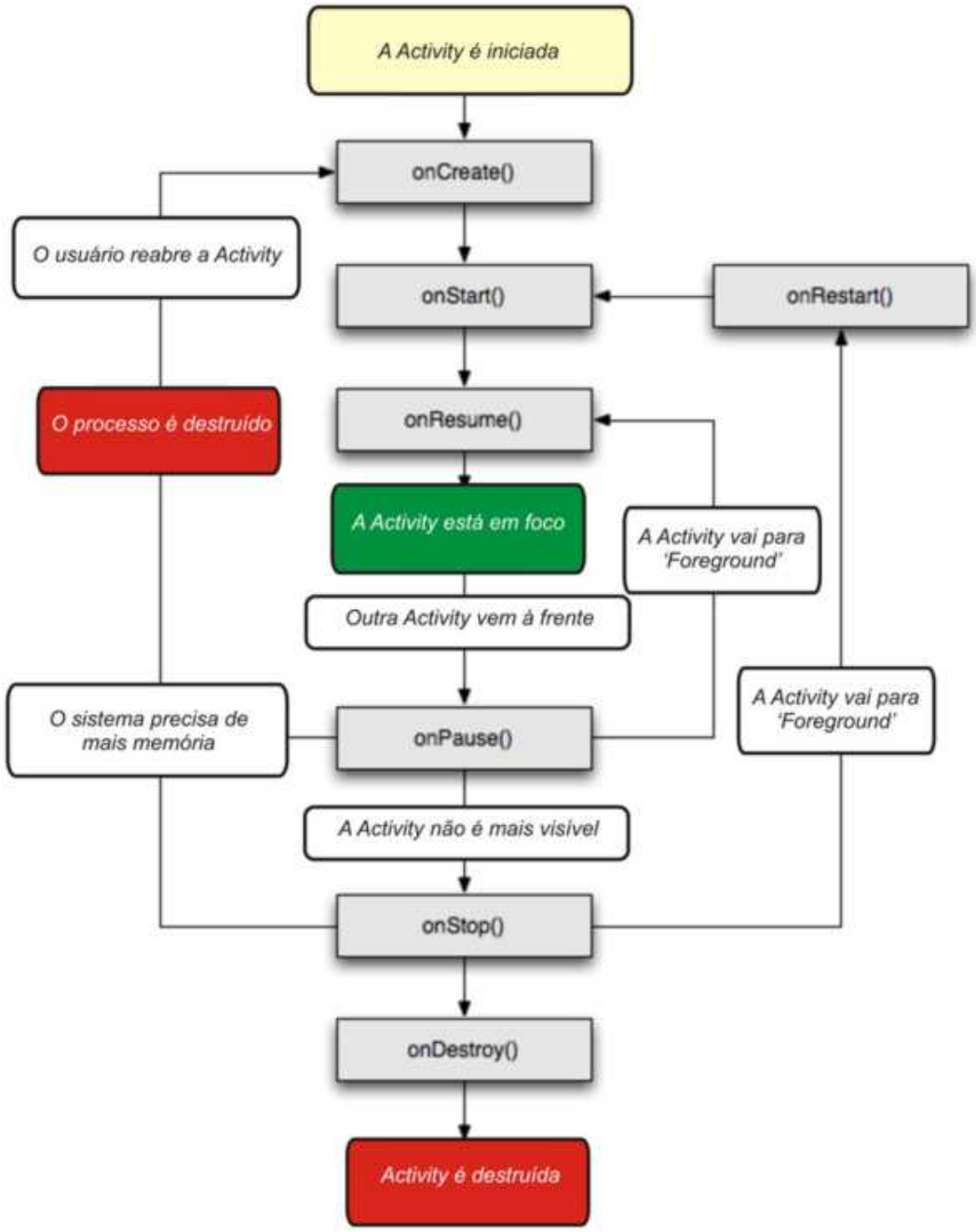

Figura 5: Ciclo de vida de uma Activity.
Fonte: felipesilveira.com.br.

uma ou mais *activitys* e, uma vez que elas tratam da interface com o usuário, cada tela do aplicativo possui uma *activity* associada a ela.

Portanto, para entender como um aplicativo funciona é essencial introduzir alguns conceitos relacionados as *activitys*. O ciclo de vida de uma *activity* pode ser observado na Figura 5.

Existem dois métodos principais que todas as *activitys* devem tratar, são eles:

- ❏ onCreate(Bundle): É por onde cada tela é iniciada, nesse momento deve ser in-

formado qual o *layout* está associado à essa *activity*. Nas aplicações Android, um *layout* é um arquivo XML que define todos os objetos gráficos contidos em uma tela.

- ❑ onPause(): Nesse método são tratadas as atividades necessárias para o usuário sair dessa tela, as informações pertinentes devem ser salvas ou transferidas para a próxima *activity*.

A classe *activity* é, portanto, uma parte importante do ciclo de vida total de um aplicativo, ela determina a forma e o fluxo como as atividades são executadas e colocadas juntas.

Capítulo 3

Materiais e Métodos

A partir dos conceitos discutidos na seção 2.1, foi criado um modelo de arquitetura de um sistema e um protótipo, a fim de validar o modelo e testar o funcionamento do sistema como um todo.

Nesse capítulo será discutido os detalhes de desenvolvimento do protótipo em cada esfera do sistema.

3.1 Materiais

Nessa seção serão apresentados os materiais utilizados no desenvolvimento, como bibliotecas de *software* e componentes de *hardware*.

3.1.1 Ruby

A linguagem de programação utilizada para criação da Aplicação Web, o Ruby, foi desenvolvida no Japão por Yukihiro Matsumoto, conhecido como "Matz", em 1993. Devido ao seu grande interesse e por acreditar que as linguagens *scripts* seriam o futuro da programação, como não existia nenhuma linguagem com as características que ele buscava, decidiu desenvolver a própria linguagem. Matz se concentrou em desenvolver uma linguagem de programação que tinha como foco as necessidades humanas ao invés dos computadores, por isso é uma linguagem de leitura e compreensão muito fáceis. (STEWART, 2001)

O Ruby é uma linguagem alto nível, ou seja, uma linguagem com grande nível de abstração, mais próxima da linguagem humana do que da linguagem de máquina. Também é uma linguagem de programação interpretada, não necessita, portanto, de um compilador, ela é executada pelo interpretador, um programa instalado com essa finalidade e só depois é executada pelo sistema operacional ou processador. Orientada a Objetos, significa que ela permite aos usuários manipularem estruturas de dados, chamadas objetos, para criar e executar programas, no Ruby todos os elementos são objetos. (CLINTON, 2009)

3.1.2 Ruby-on-Rails

Para o desenvolvimento *web*, foi utilizado um *framework*, conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema de aplicação (FAYAD; SCHMIDT, 1997) no modelo *Model-view-controller* (MVC). O *framework* escolhido foi o *Ruby-on-Rails*, criado por David Heinemeier Hansson, em 2004. Um projeto *open-source* escrito na linguagem de programação *Ruby* e otimizado para produtividade, facilidade e rapidez na implementação de aplicações *web* orientadas a banco de dados.

O modelo MVC é utilizado pelo Ruby-on-Rails para organizar a programação da aplicação. Em uma configuração padrão do *Rails*, um *Model* se refere a uma tabela no banco de dados e um arquivo Ruby, por exemplo, um modelo de *User*, será definido em um arquivo user.rb, na pasta específica dos modelos e estará relacionado com a tabela *users* no banco de dados, existe a possibilidade de escolher outros nomes e fazer outros relacionamentos, como relacionar a tabela clientes ao modelo *User*, mas não é boa prática e nem segue a filosofia de *Convention over Configuration* (CoC). Os *models*, *views* e *controllers* do sistema desenvolvido serão listados e explicados posteriormente.

Entre as principais características do Ruby-on-Rails estão os conceitos de *Don't Repeat Yourself* (DRY) e CoC. O primeiro, faz referência à ideia de reutilização de código, que também é uma das principais vantagens e características da orientação a objetos, o próprio *framework* Ruby-on-Rails nos incentiva a adoção de padrões de projeto mais adequados com essa finalidade. O segundo conceito, CoC, traz o benefício de poder escrever menos linhas de código para implementar determinada funcionalidade, essa facilidade no desenvolvimento vem com o preço de o sistema precisar seguir alguns padrões como, por exemplo, padrões específicos para nomear os arquivos, classes e métodos, e também devem estar em pastas com nomes específicos, entre outras regras. (CAELUM, 2004)

O Rails é um *meta-framework*, ele foi desenvolvido a partir de outros *frameworks*, portanto para funcionar algumas dependências precisam ser instaladas junto ao pacote Ruby-on-Rails, são elas:

- *ActionMailer*: *framework* que permite enviar e receber emails a partir da aplicação usando uma classe específica, *ActionMailer*, que é muito similar a um *controller*.

- *ActionPack*: é um *framework* que trata e responde as requisições *web*, também prove mecanismos de roteamento, mapeando as requisições para os *controllers* que implementam as ações e, com os dados fornecidos são responsáveis por renderizar os *views*. Esse *framework* é o responsável pelo *view* e *controller* do modelo MVC.

- *ActionView*: é um *framework* para manipular os *views*, pesquisa de *templates* e renderização, também oferece uma estrutura que ajuda a criar formulários HTML. Os *templates* tem formato .ERB(*embedded Ruby*, usados como pequenos trechos de Ruby dentro do HTML).

- *ActiveJob*: é o *framework* utilizado para declarar os trabalhos e faze-los rodar em várias filas *backends*. Esses trabalhos podem ser várias coisas, entre elas limpezas regulares, emails, ou praticamente qualquer tarefa que possa dividida em pequenas unidades de trabalho e rodar paralelamente.

- *ActiveModel*: fornece um conjunto de interfaces para ser utilizada nos modelos e classes. Esse *framework* permite um objeto interagir com o *ActionPack* usando os módulos de um modelo diferente do seu próprio.

- *ActiveRecord*: é um padrão de software encontrado em programas que utilizam banco de dados relacionais, todo objeto deve incluir funções como inserir, editar e apagar. Banco de dados relacionais modelam os objetos como tabelas, e definem relações entre as tabelas. Esse módulo mapeia as tabelas nas classes Ruby.

- *ActiveSupport*: é uma coleção de classes úteis e extensões da biblioteca padrão utilizada no desenvolvimento do Ruby-on-Rails.

- *Bundler*: fornece um ambiente consistente de projetos Ruby, ele rastreia e instala as *gems* necessárias em suas versões especificadas para que o projeto possa ser executado.

- *RailTies*: responsável por juntar todos os *frameworks* utilizados. Cuida do processo de inicialização de uma aplicação Rails e gerencia a interface que disponibiliza uma linha de comando Rails.

- *sprockets-Rails*: é uma biblioteca que compila e cuida dos *web assets*. Ele fornece um gerenciador de dependências para JavaScript e CSS, e também um poderoso pré-processamento que permite ao desenvolvedor escrever *scripts* em linguagens como CoffeeScript, Sass, SCSS e LESS.

- *ActiveResource*: representam os recursos RESTful para manipular objetos Ruby. Para mapear os recursos dos objetos que fazem referência, o Ruby só precisa que o nome da classe seja correspondente ao nome do recurso, por exemplo a classe Usuário estende seus recursos aos objetos usuários.

3.1.3 Biblioteca SPI

Serial Peripheral Interface (SPI) é um protocolo de comunicação serial síncrono, utilizado pelos microcontroladores para se comunicarem com outros dispositivos periféricos, a curta distância, de maneira rápida, também pode ser usado na comunicação entre dois microcontroladores.

Em uma conexão SPI sempre existe um *master*, normalmente um microcontrolador, e um *slave* que são os dispositivos periféricos a serem controlados, um *master* é capaz de

comunicar-se com vários *slaves*, normalmente três ligações são comuns entre o *master* e todos os *slaves*, são elas:

- *Master In Slave Out* (MISO): o canal para o *slave* enviar dados ao *master*.
- *Master Out Slave In* (MOSI): o canal para o *master* enviar dados para o *slave*
- *Serial Clock* (SCK): os pulsos de *clock* que são utilizados para sincronizar a transmissão dos dados, são gerados pelo *master*.

Além dessas, existe uma ligação específica que o *master* utiliza para selecionar com qual *slave* deseja se comunicar, essa ligação é única para cada *slave* e é chamada de *Slave Select* (SS).

No sistema desse trabalho de conclusão de curso, essa biblioteca é utilizada tanto pela Central de Controle quanto pelos Periféricos para que o microcontrolador ATmega328 consiga comunicar-se com os módulos *wireless* nRF24L01.

3.1.4 Biblioteca RF24

Essa biblioteca foi desenvolvida especificamente para utilizar o Arduino como um *driver*, ou seja, o *master* na conexão com o módulo nRF24L01, o transceptor de 2.4GHz utilizado no projeto. Foi criada com os objetos de maximizar a conformidade com as operações pretendidas pela Nordic, criadora do chip nRF24L01+, facilitar a utilização por usuários iniciantes e fornecer uma *interface* similar às bibliotecas padrões do Arduino.

Os principais métodos, necessários para operar o *chip* da Nordic são os seguintes:

- *Begin*: processo de inicialização do *chip*, deve ser chamada no *setup*.
- *startListening*: começa a "ouvir", esperar por dados, nos canais abertos para leitura.
- *stopListening*: para de "ouvir" os canais de comunicação.
- *write*: envia uma mensagem no duto de comunicação aberto para escrita.
- *available*: verifica se existem dados disponíveis para serem lidos.
- *read*: lê o primeiro dado disponível no *buffer*.
- *openWritingPipe*: abre um duto de comunicação em um endereço específico para envio de dados.
- *openReadingPipe*: abre um duto de comunicação em um endereço específico para recebimento de dados.

Essa biblioteca também possibilita configurações nos módulos nRF24L01, é possível definir especificações como o tamanho, em bits, dos dados que serão enviados através dos módulos nRF24L01, quantidade de tentativas de reenviar os pacotes de mensagens antes de retornar uma mensagem de que o dado foi perdido, entre outros.

3.1.5 Biblioteca Bridge

Uma vez que a Central de Controle conta com um Arduino UNO conectado a um Draginio Yún Shield, ela possui um microcontrolador, o ATmega328, que roda um *sketch* Arduino e um *System-on-a-chip* (SoC) AR9331 (ATHEROS, 2010), que roda um SO com kernel Linux, baseado no OpenWRT (https://openwrt.org). É possível chamar programas ou *scripts* customizados do lado Linux através da *sketch* Arduino rodando no microcontrolador e, além disso, acessar vários serviços de internet.

A biblioteca Bridge.h é responsável por facilitar a comunicação entre o SoC AR9331 e o microcontrolador ATmega328. Ela herda de uma biblioteca chamada Stream.h, que possui vários métodos de comunicação Serial.

Os comandos utilizados através da biblioteca Bridge.h são interpretados pelo Python no AR9331. Sua função é executar programas no lado do GNU/Linux quando solicitado pelo Arduino, também fornece um espaço para armazenamento compartilhado de dados como leitura de sensores entre o Arduino e a Internet, também é responsável por receber comandos da internet e passar diretamente para o lado Arduino. A biblioteca oferece uma comunicação *full-duplex*. (ARDUINO, 2015)

A Bridge.h tem uma classe chamada HttpClient, essa classe é uma extensão da biblioteca Process.h e atua como um *wrapper* para os comandos de cURL, criando um cliente HTTP do lado Linux. Um objeto dessa classe é utilizado pelo lado do Arduino para fazer requisições HTTP para o servidor.

3.1.6 Hardware

Nesta seção, serão apresentados, em maior nível de detalhamento, os componentes de *hardware* utilizados até o momento.

3.1.6.1 Arduino UNO

Arduino é uma plataforma de prototipagem eletrônica de placa única e *hardware* livre, possui uma linguagem de programação padrão, a qual tem origem em Wiring, e é essencialmente C/C++. O objetivo do projeto é criar ferramentas que são acessíveis, com baixo custo, flexíveis e fáceis de usar. O projeto iniciou-se na Itália, em 2005, com o objetivo de desenvolver uma plataforma de prototipagem barata para projetos escolares. (ARDUINO, 2014c)

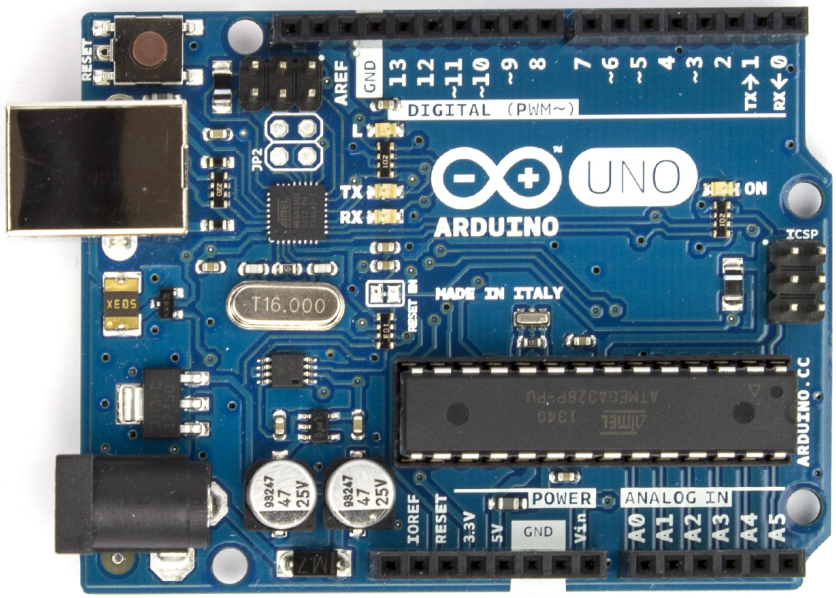

Figura 6: Arduino Uno.

Fonte: arduino.cc.

Nesse projeto, para a construção do protótipo, será utilizada a placa Arduino Uno R3, tanto para o protótipo da Central de Controle quanto para o protótipo dos Periféricos. A escolha dessa placa de prototipagem foi baseada em pesquisas dos parâmetros e funcionalidades que atendessem aos requisitos do sistema, como capacidade de processamento e memória, também foi levado em consideração preço, disponibilidade e facilidade de acesso, assim como a facilidade de embarcar o sistema.

O Arduino Uno R3 é uma placa microcontroladora de prototipagem que tem como base o chip ATmega328 que possui 14 entradas/saídas digitais, das quais 6 podem ser utilizadas como *Pulse-Width Modulation* (PWM), 6 entradas analógicas, um cristal oscilador de 16Mhz, conexão *Universal Serial Bus* (USB) através do ATmega16u2 programado como conversor USB-Serial, entrada para fonte, soquetes para *In Crcuit Serial Programming* (ICSP) e um botão de reset. (ARDUINO, 2014b)

"Uno" significa, um em italiano, foi nomeado dessa forma para marcar o lançamento do Arduino 1.0, o Uno e a versão 1.0 serão as versões de referência do Arduíno, que atualmente encontra-se na revisão 3, revisão essa que será utilizada no protótipo.

3.1.6.2 Dragino Yún Shield

Um dos fatores determinantes para a enorme versatilidade e popularidade da plataforma Arduino são os *shields*, que são placas desenvolvidas com um layout específico de modo que possam ser plugadas no topo das placas Arduino, adicionando funcionalidades e estendendo a capacidade das placas Arduino. (ARDUINO, 2014a)

Para a central de controle, em conjunto com o Arduino Uno, será utilizado o *shield* Dragino Yún Shield (Figura 7), um dos mais poderosos *shields* para placas Arduino, foi desenvolvido para resolver os problemas de conectividade e armazenamento dos microcontroladores utilizados nas placas Arduino (DRAGINO, 2014). O Yún Shield possui um processador de 400Mhz AR9331, 16MBytes de memória *flash* e 64MBytes de *Random Access Memory* (RAM), um conector RJ45, conexão WiFi 802.11 b/g/n com uma antena externa via um conector I-Pex, um botão de reset independente da placa Arduino onde o *shield* está sendo utilizado e um conector USB *host*.

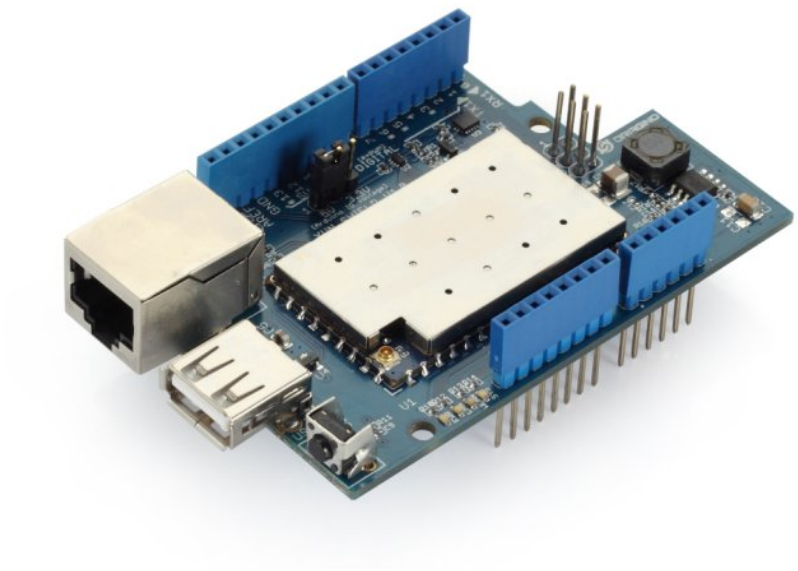

Figura 7: Dragino Yún Shield.
Fonte: www.dragino.com.

O Dragino Yún Shield é um SoC que roda um sistema operacional baseado no *kernel* Linux, o OpenWRT, largamente utilizado em sistemas embarcados para rotear tráfego de rede, pode ser configurado utilizando uma interface de linha de comando *ash shell* ou através de uma *Graphic User Interface* (GUI) chamada LuCI. A placa é compatível com a IDE Arduino, onde é possível fazer o *upload* dos programas para a placa Arduino onde o Yún Shield está conectado através da rede WiFi, o Yún Shield transfere o programa recebido por WiFi para a placa Arduino através de uma comunicação SPI e pode ser configurado através de uma Web GUI, *Secure Shell* (SSH), *Local area network* (LAN) ou WiFi.

Outro fator importante é que a comunicação com a internet, seja através da rede WiFi ou WAN através do conector RJ45, é feita no *core* do Dragino Yún Shield, este, por sua vez, transfere os dados tratados para o microcontrolador, essa estrutura está representada pela Figura 8.

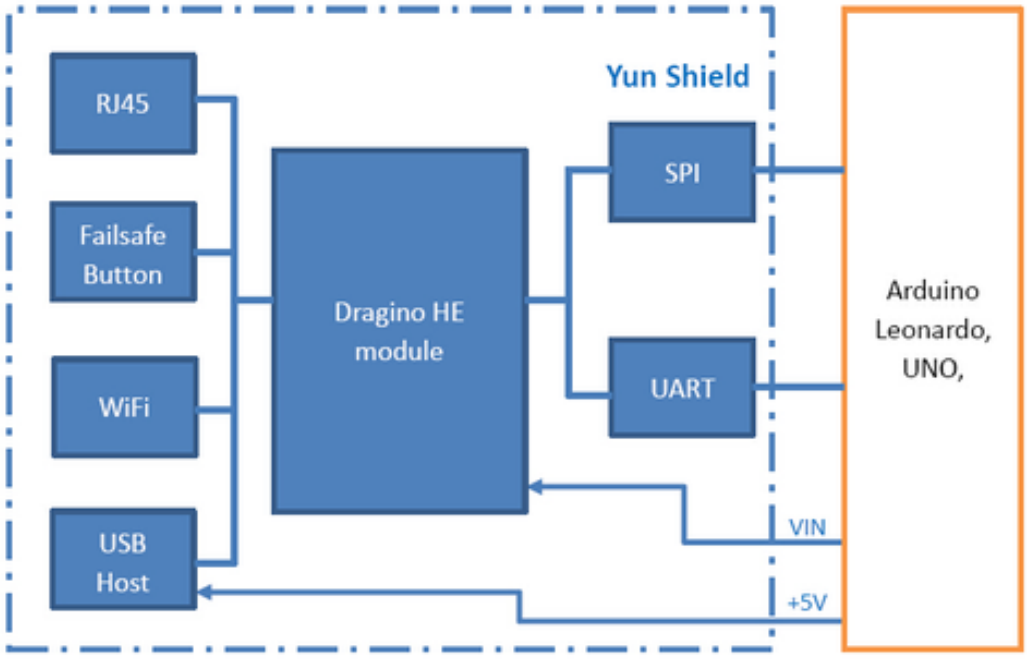

Figura 8: Dragino Yún Shield Struct.
Fonte: www.dragino.com.

3.1.6.3 Módulo nRF24L01+

A comunicação entre a Central de Controle o os Periféricos é feita através de um módulo *wireless transceiver* que tem como base o chip nRF24L01+, fabricado pela Nordic. Esse módulo foi escolhido devido ao tamanho, baixo consumo de energia, preço e facilidade de implementação com o Arduino, uma vez que existe uma biblioteca que disponibiliza opções de configurações e funcionalidades para estabelecer uma comunicação entre módulos desse tipo.

O Nordic nRFL01+ é um transceptor de radiofrequência que opera em 2.4Ghz, é altamente integrado e possui um consumo de energia muito baixo. Os picos de correntes de transmissão e recepção são menores que 14mA, possui um gerenciamento avançado de energia e a faixa de alimentação é de 1.9 a 3.6V, sendo então possível alimentar o módulo através do pino que oferece 3.3V do Arduino, essa fonte integrada na placa Arduino pode fornecer uma corrente de até 50mA (ARDUINO, 2014b). O módulo, da fabricante Nordic, fornece uma solução *Ultra Low Power* (ULP) verdadeira, permitindo de meses a anos de vida útil se alimentados com pilhas do tipo AA/AAA (SEMICONDUTOR, 2014). O Nordic nRF24L01+ possui uma solução completa para transmissão e recepção de dados à uma frequência de 2.4 Ghz, possui sintetizador *Radio Frequency* (RF) e lógica baseband, incluindo o Enhanced ShockBurst e protocolo de hardware para uma interface SPI de alta velocidade para o controlador da aplicação. Não é necessário nenhum loop filter, ressonadores, diodos ou capacitores externos, apenas um cristal de baixo custo, circuitos

Figura 9: Módulo nRF24L01+.
Fonte: www.filipeflop.com/.

de correspondência e uma antena. (SEMICONDUTOR, 2014)

Enhanced ShockBurst (ESB) é um protocolo básico que suporta duas vias de comunicação através de pacotes de dados, possui algumas características como *Packets Buffering*, capaz de enfileirar os pacotes a serem enviados, liberando o dispositivo controlador para outras tarefas, também possui verificação dos pacotes, que é capaz de assegurar a integridade dos pacotes recebidos e retransmissão automáticas de pacotes perdidos, essa funcionalidade é configurável para o desenvolvedor, é possível determinar um número específico de tentativas de retransmissão automática.

O módulo escolhido (Figura 9), acompanha a antena, o cristal e os circuitos de correspondência para uma transmissão na frequência de 2.4GHz, em uma velocidade de operação de até 2Mbps, modulação *Gaussian Frequency-Shift Keying* (GFSK), habilidade de anti-interferência, verificação de erros por *Cyclic Redundancy Check* (CRC), comunicação multi-ponto de 125 canais e controle de fluxo e regulador de tensão embutido.

A escolha desse módulo, ao invés dos tradicionais transmissores e receptores RF, foi devido ao fato que, como a maioria dos Periféricos irão possuir algum tipo de *feedback*, e o protocolo de comunicação, oferecido pela biblioteca RF24, utilizado por esse módulo é do tipo *handshaking*, ou seja, o dispositivo que é o *master* toma a iniciativa da comunicação com um *slave*, nesse projeto o *master* será sempre a Central de Controle, e então espera por uma resposta em um canal de comunicação específico para confirmar que o dado foi enviado, as leituras dos sensores instalados nos Periféricos, que são os *slaves*, serão enviadas para a Central de Controle através dessa resposta, quando um pacote de dados for recebido.

3.2 Métodos

Nessa seção são mostrados os modelos criados, que foram baseados no projeto IoT-A, para o sistema em questão e todos os detalhes de implementação do protótipo desenvolvido utilizado para validar o modelo proposto.

3.2.1 Modelagem do Sistema

Esse trabalho tratou do caso de uso mais simples do sistema proposto, que pode ser estendido para sistemas maiores e mais complexos utilizando-se da mesma arquitetura. Esse caso consiste de uma pessoa que deseja controlar uma lâmpada de sua residência, remotamente, através do seu *smartphone* e verificar o estado da mesma, ou seja, se está acesa ou apagada.

3.2.1.1 Modelo de Domínio

Segundo o projeto IoT-A e suas recomendações com relação à modelagem de sistemas, o Modelo de Domínio é o ponto de partida para projetar um sistema com a mentalidade de Internet das Coisas. A Figura 10 é o modelo desenvolvido, para esse projeto, que trata o caso de uso apresentado.

Nesse modelo estão representados os serviços que o sistema deve oferecer. Essa representação oferece uma visão com alto nível de abstração, porém agrega aspectos mais próximos a uma implementação. Fica explícito todas as relações entre os componentes do sistema e quais são os elementos de *hardware* e *software*, mais detalhes sobre o significado dessas relações foram apresentados na Seção 2.1.

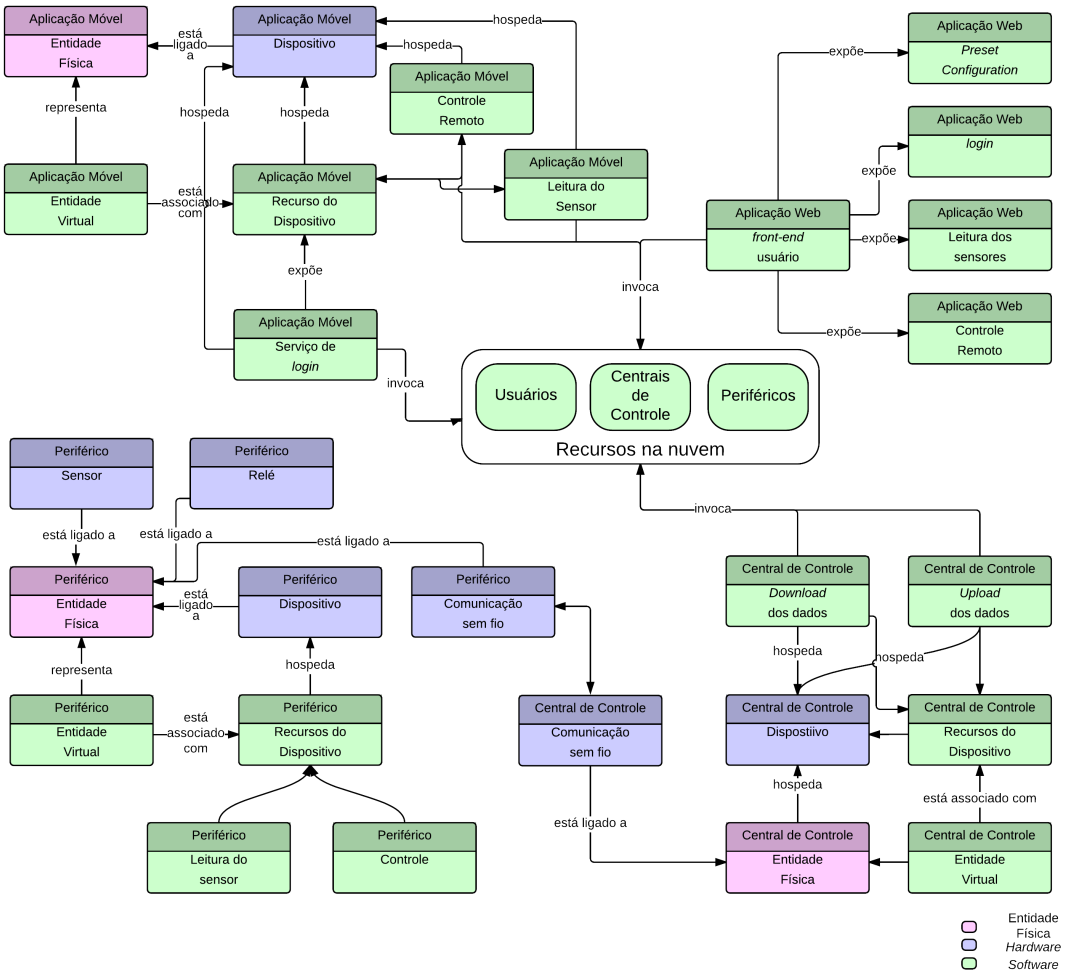

Figura 10: Modelo de Domínio do Sistema.

3.2.1.2 Modelo de Informação

O Modelo de Informação auxilia no *design* do sistema, evidenciando as entidades de banco de dados, valores, atributos, metadados e suas interações com os serviços e recursos oferecidos pelo sistema. Esse modelo aproxima os conceitos tratados no Modelo de Domínio, Figura 10, à implementação propriamente dita. Dessa forma, as entidades físicas, que agora estão representadas no mundo virtual através de entidades virtuais, são destacadas como sendo o ponto principal de um sistema voltado à Internet das Coisas.

Os Modelos de Informação das Entidades Virtuais Central de Controle e Periférico estão representados, respectivamente, pelas Figura 11 e Figura 12.

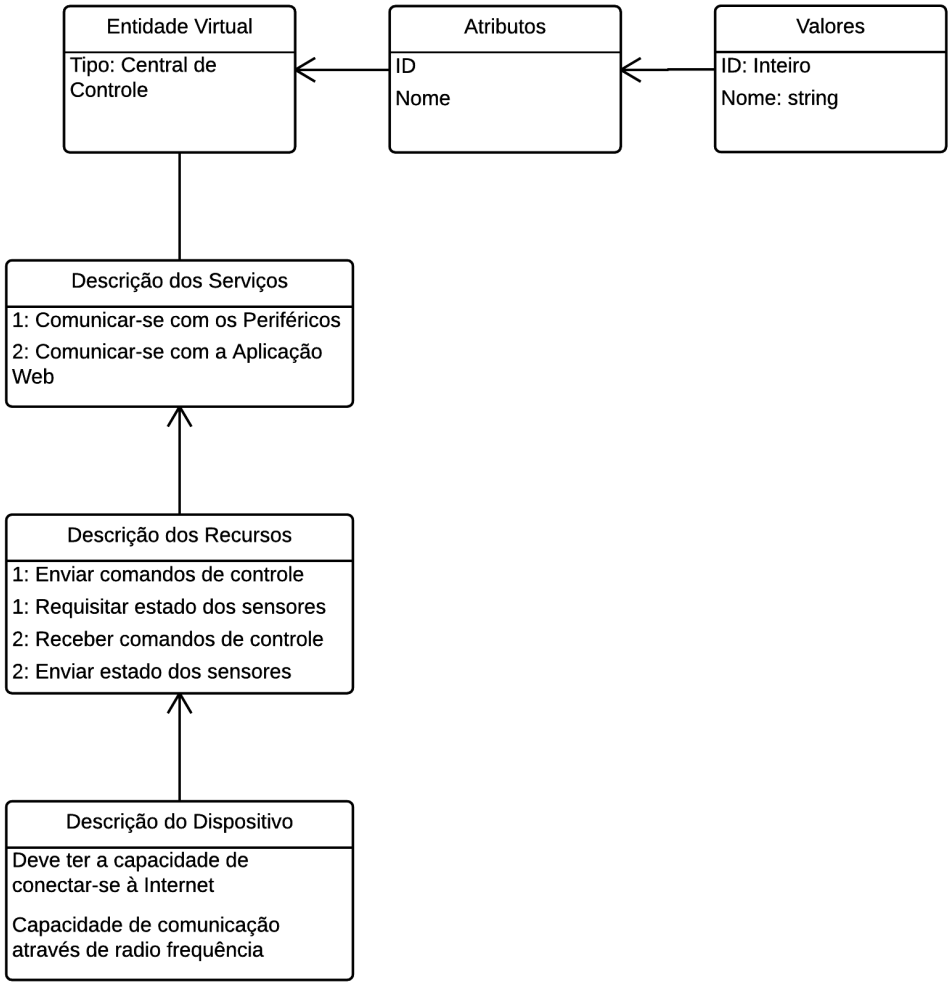

Figura 11: Modelo de Informação do Controlador.

Nessa seção foram apresentados os Modelos de Domínio e Informação do sistema proposto por esse trabalho de conclusão de curso. A seguir cada componente do sistema, que pode ser visualizado na modelagem de domínio, será tratado individualmente para, posteriormente, desenvolver um protótipo do sistema proposto que valide o mesmo.

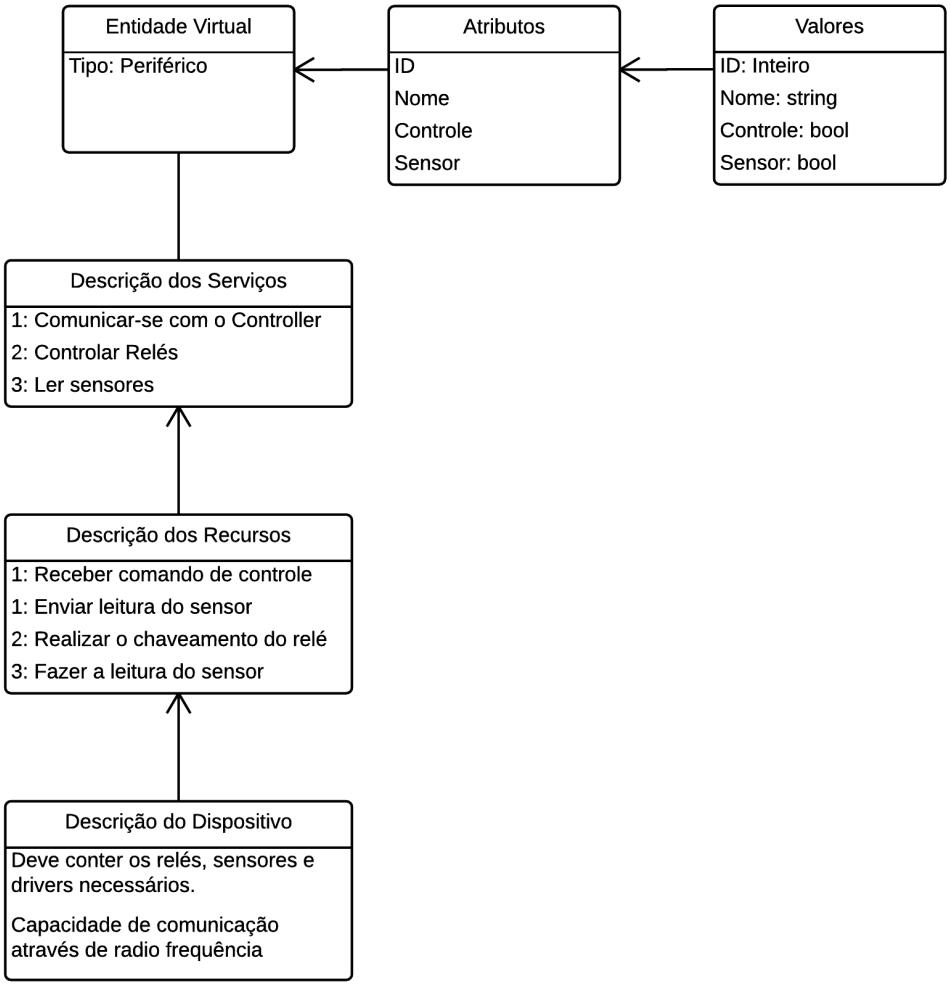

Figura 12: Modelo de Informação do Periférico.

3.2.2 Componentes Do Sistema

A partir dos modelos apresentados na seção anterior, foi possível chegar em uma arquitetura específica para o sistema desse trabalho de conclusão de curso. Na presente seção os componentes do sistema, que compõe a arquitetura definida, serão explicados detalhadamente, apresentando as características específicas de cada um, alguns conceitos envolvidos e os requisitos mínimos de cada parte do projeto, assim como os detalhes de implementação de um protótipo que visa validar o modelo proposto. Uma visão geral do sistema pode ser vista na Figura 13.

O caso de uso tratado é de um usuário utilizando a Aplicação Web para cadastrar sua Central de Controle e Periféricos, dispositivos os quais foram instalados fisicamente no

local onde deseja-se controle remoto. A partir deste momento, ele é capaz de controlar remotamente seus Periféricos, através das interfaces disponíveis na própria Aplicação Web ou através da Aplicação Móvel instalada em seu *smartphone*. A Central de Controle é quem faz a ponte de comunicação entre os Periféricos e à Aplicação Web, quando um comando é dado, por exemplo, através do aplicativo, este envia, primeiramente, o comando para a Aplicação Web e ela é quem transmite o comando, através da internet, para a Central de Controle que, por sua vez, encaminha por uma comunicação sem fio para o Periférico responsável.

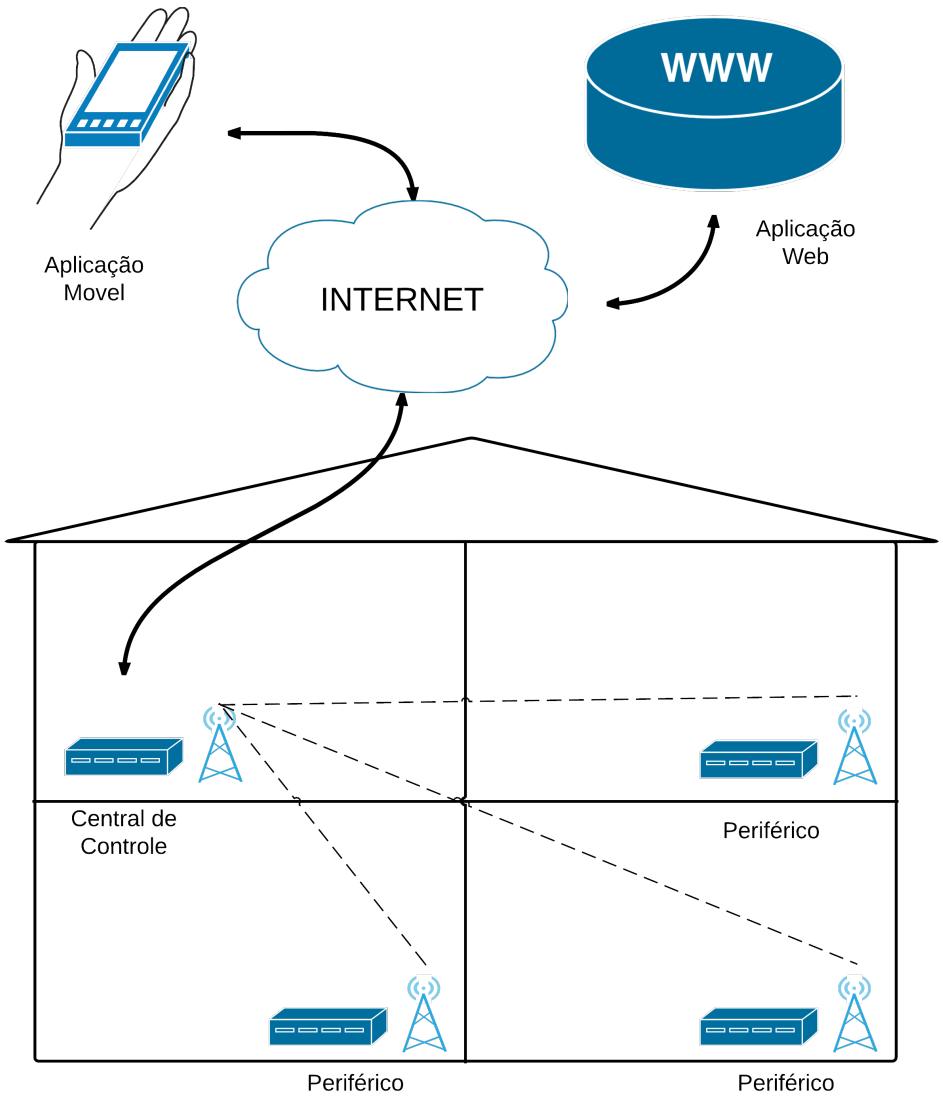

Figura 13: Visão Geral Do Projeto.

A seguir, cada componente do sistema é tratado individualmente e são apresentados os detalhes de implementação do protótipo.

3.2.3 Aplicação Web

A Aplicação Web tem três principais funções: (1) Oferecer um *front-end* para os usuários cadastrarem suas centrais de controle e periféricos, com interfaces que permitem que o mesmo gerencie seus dispositivos e controle-os. (2) Uma API para a Aplicação Móvel, que disponibiliza as informações necessárias para que o aplicativo seja capaz de mostrar para um usuário os dispositivos que controla e, de fato, controlá-los através de seu *smartphone*. (3) Uma API para a Central de Controle, disponibilizando para a mesma as informações sobre quais os periféricos estão cadastrados.

Um dos requisitos da Aplicação Web é que a mesma esteja hospedada em um servidor que forneça uma infra estrutura suficientemente grande para que, caso haja necessidade, possa contratar mais recursos. Uma vez que toda a comunicação entre o usuário e os Periféricos passa pela Aplicação Web, quanto mais usuários utilizando o sistema, mais Centrais de Controle instaladas, mais requisições simultâneas ao servidor, portanto, ele deve ser capaz de tratar todas essas concorrências de requisições de forma a não impactar negativamente na experiência do usuário. A decisão por não hospedar a Aplicação Web na própria Central de Controle estão discutidas no capítulo 6.

Para o desenvolvimento da Aplicação Web, faz muito sentido que a mesma seja desenvolvida com uma arquitetura de *software* no modelo MVC, uma vez que o *front-end* do usuário possui um *layout* padrão em todas as telas, mas diferentes usuários possuem diferentes Centrais de Controle e Periféricos. Para ficar mais claro o modelo MVC, na subseção seguinte será apresentada uma breve explicação sobre os conceitos do modelo. Esses conceitos serão necessários para entender o desenvolvimento do protótipo. Logo em seguida, uma breve explicação sobre oque é uma API.

A Aplicação Web foi desenvolvida utilizando a linguagem de programação Ruby, em conjunto com o *framework* Ruby-on-Rails, hospedado no servidor Heroku, um *platform as a Service* (PaaS). A seguir uma breve explicação sobre as características específicas desses componentes e, logo após, os detalhes de implementação da Aplicação Web do protótipo.

3.2.3.1 Desenvolvimento

Após a apresentação de alguns conceitos importantes sobre o *framework* utilizado, a linguagem de programação e os conceitos do modelo MVC e API do Capítulo anterior, nessa seção é detalhado os por menores do desenvolvimento da Aplicação Web.

Na estrutura de um projeto desenvolvido utilizando o Ruby-on-Rails, existem alguns arquivos importantes para a compreensão do sistema, o primeiro a ser detalhado será o routes.rb. Esse é o arquivo que mapeia as requisições HTTP e redirecionam para os *controllers* correspondentes e também é aqui onde os recursos disponíveis estão definidos, como os *users*, *peripherals* e *devices* que serão explicados na seção sobre os *models* do sistema.

A primeira definição é a root, ou seja, quando nenhuma URI é fornecida, esse tipo de requisição é feita por um usuário acessando o sistema pelo seu navegador, ele é então redirecionado para um *view* onde é possível fazer o *login* ou criar um novo usuário. Esse padrão se repete para todo o sistema do Ruby-on-Rails, a requisição é recebida no route.rb, que tem a tarefa de acionar um *controller*, esse é o responsável por receber os parâmetros da URI, casos existam, e, utilizando esses parâmetros, acionar um *view* e/ou realizar alguma alteração em algum *model*.

Portanto, exemplificando, quando um usuário acessa o site, o controller *sessions* é acionado, e a viewindex é exibida ao usuário. Nessa página é possível o usuário fazer um *login* ou criar um novo *User*, aqui *User* faz referência a um *model* do sistema, e o *controller Session* é responsável por buscar um *User* no *model* ou criar um, caso essa ação seja solicitada. Esse fluxo pode ser reproduzido para toda Aplicação Web, portanto, para explicar o sistema, será necessário apenas explicar os *Models*, *Views* e *Controllers*.

3.2.3.2 Modelo

Os *Models*, são as estruturas de dados, no Ruby-on-Rails são tabelas em um banco de dados, as tabelas e as relações entre elas podem ser vistos na Figura 14.

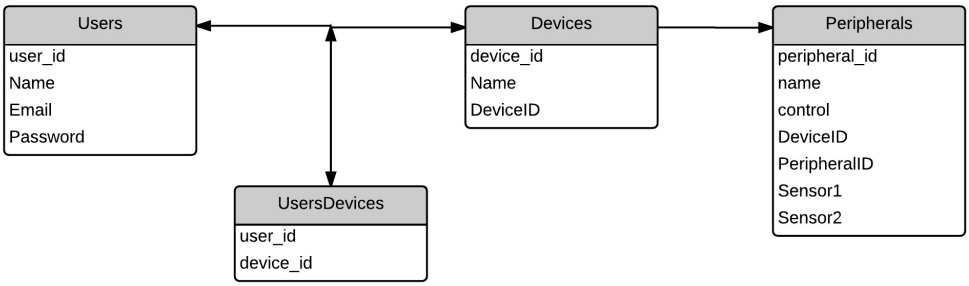

Figura 14: Modelo do Banco de Dados.

O modelo *Users* são usuários do sistema, pessoas que se cadastraram no site, elas possuem um nome, e-mail e um *password*. Esses dados identificam o usuário ao sistema.

Devices são as Centrais de Controle, elas possuem um nome e um DeviceID. As Centrais de Controle são criadas por usuários cadastrados no sistema, eles podem inserir um nome que identifica o lugar onde a Central de Controle foi instalada, por exemplo, Escritório ou Minha Casa, e precisam inserir um DeviceID, que é um identificador único dessa Central de Controle, trata-se de um valor fixo determinado pelo fabricante.

UsersDevices é um modelo auxiliar, uma vez que a relação entre usuários e Centrais de Controle é uma relação nxn, ou seja, um mesmo usuário pode ter uma ou mais Centrais

de Controle e uma Central de Controle pode ter um ou mais usuários, uma tabela auxiliar que guarde a informação de qual *device* está associado a qual *user* é necessária.

Peripherals representam os periféricos instalados pelo usuário do sistema, esses dispositivos precisam ser instalados fisicamente no local onde deseja-se acionamento remoto, também precisam ser cadastrados no sistema *web*. Eles possuem um nome, que identificam o periférico para o usuário como, por exemplo, Lâmpada da Sala ou Portão da Garagem, possuem três campos booleanos chamados de *control*, *sensor1* e *sensor2*, esses campos identificam, respectivamente, uma variável para o controle do periférico e a leitura de dois sensores que podem ou não estar presentes nos mesmos. Possuem um campo chamado DeviceID, que associa esse Periférico a uma Central de Controle, todo periférico precisa estar contido em uma Central de Controle e, por último, um campo chamado PeripheralID, também é um valor fixo determinado pelo fabricante do periférico que precisa ser inserido manualmente pelo usuário.

3.2.3.3 Views

Os *views* são as páginas que os usuários navegam, ou os dados que são enviados à Central de Controle ou a Aplicação Móvel através de sua API, cada *model* está associado a pelo menos um *view*.

Os usuários que utilizam o sistema, podem navegar através das páginas do servidor *web* e cada ação está associada a um *view*, entre as ações disponíveis para eles estão criar, editar, visualizar ou excluir seu próprio *user*, qualquer *device* ou *peripheral* que o mesmo tenha acesso. Através dos *views* os usuários também podem verificar o estado dos sensores instalados em cada periférico e controlar remotamente os mesmos.

3.2.3.4 Controlador

O sistema possui sete *Controllers*, a seguir uma explicação mais detalhada das funções de cada controlador:

1. *ApplicationController*: esse controlador da aplicação contém somente uma função, chamada de *CurrentUser* responsável por criar um objeto do tipo *user* para a sessão caso o usuário faça *login* no site.

2. *DevicesController*: responsável por qualquer ação relacionada ao modelo *device*, ele possui funções básicas para criar, editar, atualizar e destruir objetos do tipo *device*, e além das funções básicas, possui a função *index* para listar todos os devices relacionados ao objeto *CurrentUser*, retornado do controlador *ApplicationControler*, e *show*, um método para mostrar um *device* específico que o usuário deseja.

3. *PeripheralsController*: responsável por qualquer ação relacionada ao modelo *peripherals*, ele possui funções básicas para criar, editar, atualizar e destruir objetos

do tipo *peripherals* e, além das funções básicas, possui a função *index* para listar todos os periféricos relacionados a um *device*, e *show*, um método para mostrar um periférico específico que o usuário deseja verificar o estado dos sensores ou controlar o mesmo.

4. *SessionsController*: possui três funções, *login*, que utiliza outra função do mesmo controlador chamada de *authorize*, que busca por uma combinação de e-mail e senha na tabela do banco de dados referente aos *Users*, caso a função encontre essa combinação, retorna o *userID* do usuário, e a função *logout* que encerra a sessão fazendo com que o *userID* seja nulo.

5. *TasksController* Esse controlador contém todas as funções da API utilizada pelo navegador, Central de Controle ou Aplicação Móvel, ao todo são sete funções:

   getInfo utilizada pela Central de Controle para receber todos os dados de todos os periféricos de um DeviceID específico, passado como parâmetro.

   updateSensor, função utilizada pela Central de Controle quando a mesma deseja alterar o valor de um sensor, os parâmetros necessários são DeviceID, PeripheralID e qual sensor será atualizado, além disso o valor booleano da leitura do sensor.

   ControlPeripheral, é utilizada pela Aplicação Móvel ou pelo navegador para alterar o bit do periférico referente ao controle do mesmo, para isso é necessário que o parâmetro PeripheralID seja fornecido.

   mobileLogin, utilizada pela Aplicação Móvel para receber o *userID*, passando como parâmetros o e-mail e a senha, similar a função *login* do controlador *SessionsController*.

   listDevices função utilizada pela Aplicação Móvel para receber uma lista de todos os nomes e DeviceIDs associados ao *userID*, que é passado como parâmetro, recebido do método anterior chamado *mobileLogin*.

   listPeriherals função utilizada pela Aplicação Móvel para receber uma lista de todos os nomes e PeripheralIDs associados ao *deviceID*, que é passado como parâmetro, recebido do método anterior chamado *listDevices*.

6. *UserDevicesController* possui somente uma função chamada *new* para criar uma entrada na tabela UsersDevices, essa função é usada internamente para fazer uma associação de um *user* a um *device*.

7. *UsersController* possui uma função chamada *create* utilizada pelo navegador para criar o cadastro de um novo usuário.

3.2.4 Central de Controle

A Central de Controle é um sistema embarcado que faz a ponte entre os Periféricos e a Aplicação Web.

Um sistema embarcado é um sistema microprocessado no qual um computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. A diferença entre um computador comum e um embarcado, é que o último realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Uma vez que o sistema é dedicado à algumas tarefas específicas, é possível otimizar o *hardware* reduzindo seu tamanho, recursos de armazenamento e processamento e, consequentemente, o custo final do produto. (SHIBU, 2009)

Esse dispositivo precisa de capacidade de processamento suficiente para que o mesmo seja capaz de conectar-se a um servidor *web* e interpretar todos os dados de forma rápida, estável e constante.

Outra responsabilidade da Central de Controle é a comunicação *wireless* com os Periféricos, portanto o dispositivo precisa de algum tipo de comunicação com *drivers* externos como SPI, serial, USB e etc.

Para o protótipo da Central de Controle foi utilizado um Arduino Uno em conjunto com um Dragino Yun Shield e um Módulo nRF24L01+. Os detalhes de cada um desses componentes de *hardware* utilizados, e os motivos da escolha de cada um, estão apresentados na subseção 3.1.6 desse capítulo.

3.2.4.1 Desenvolvimento

Para a programação do Arduino UNO da Central de Controle, foram utilizadas as bibliotecas SPI.h, RF24.h e Bridge.h, explicadas, respectivamente, nas subseções 3.1.3,e 3.1.4 e 3.1.5.

3.2.4.2 Programa

Um fluxograma com alto nível de abstração pode ser observado na Figura 15.

Após importar as biblioteca, algumas variáveis globais utilizadas no programa são definidas. A primeira delas é o ID da central de controle, uma string de 8 bits que identifica, para a Aplicação Web, qual é a Central de Controle, essa identificação que faz com que a Aplicação Web responda à várias possíveis Centrais de Controle conectadas ao mesmo tempo no sistema, pensando em uma possível escalabilidade do projeto. Essa ID também define a URL que a Central de Controle fará suas requisições HTTP. Essa ID é uma configuração do fabricante, não é configurável e é informada ao usuário para que o mesmo cadastre esse ID na Aplicação Web. Outras variáveis também são definidas como, por exemplo, o endereço do servidor onde a Aplicação Web está hospedada.

Figura 15: Fluxograma da Central de Controle.

Também é criada uma tabela de Dados (Tabela 1), com 8 colunas e uma quantidade de linhas igual ao número de periféricos instalados na Central de Controle em específico. Essa tabela de dados armazena informações sobre a ID dos periféricos, que são os 4 bits mais significativos da string, enquanto os 4 bits menos significativos são informações sobre os sensores e controle das funcionalidades referentes à esse periférico.

Tabela 1: Exemplo de Tabela de Dados

	Dados[i]							
	0	1	2	3	4	5	6	7
Periférico 1	1	0	0	0	1	0	0	1
Periférico 2	1	0	0	0	1	0	1	1
Periférico 3	1	0	0	0	1	1	0	1

Dutos de comunicação também são definidos, esses dutos são utilizados para a comunicação, através dos módulos nRF24L01+, com os periféricos, a Central de Controle consegue receber dados através de 1 duto, com um endereço específico, e consegue trans-

mitir dados através de vários dutos, esses endereços são fixos e definidos no início da execução do programa.

Por último, são passados os parâmetros de quais pinos do Arduino Uno, estão conectados o *Chip Enable* (CE) e *Chip Select* (CS) do módulo nRF24L01+. A conexão física entre o Arduino UNO e o módulo RF24L01 está detalhada na sessão sobre os Periféricos.

Os programas Arduino, por padrão, tem duas estruturas fixas, uma chamada setup e outra loop, a estrutura *setup* é executada uma vez quando o dispositivo é ligado e, após entrar na estrutura *loop*, o programa fica em um *loop* infinito.

Na estrutura *setup*, primeiramente é executado o método *begin*, da biblioteca Bridge.h, que executa uma rotina para iniciar a comunicação do ATmega328 com o AR9331. Após feita essa inicialização, o método begin da biblioteca RF24 também é executado, nesse momento dois parâmetros, quantidade de bits das informações enviadas através dos módulos RF24L01, chamado de *Payload*, e quantidade de tentativas de enviar as mensagens antes do *timeout*, são definidos.

Na estrutura *loop*, três funções são chamadas sequencialmente, são elas: as funções *download*, enviaRF e *upload*, portanto o código fica executando essas três funções em um *loop* infinito. Abaixo segue uma explicação mais detalhada de cada uma dessas funções.

A função *download*, é a responsável por requisitar ao servidor da Aplicação Web os dados referentes à essa Central de Controle. Através de uma requisição HTTP do tipo GET, onde o parâmetro ID da Central de Controle é passado, o servidor responde com os dados referentes à essa Central de Controle. Ainda na função *download*, os dados recebidos são salvos na tabela de dados, onde cada linha corresponde a um Periférico instalado nessa Central de Controle. Um exemplo de uma tabela de dados com 3 periféricos instalados pode ser visto na Tabela 1.

Após o término da rotina de *download*, a função enviaRF é executada. Essa é a função responsável por transmitir para os Periféricos, através do módulo nRF24L01, os dados armazenados na tabela. Uma vez que cada linha de dados é referente a um Periférico, e cada um deles possui um duto de comunicação, a Central de Controle envia um pacote em cada duto para cada linha da tabela. Os dutos fazem parte da configuração do sistema, são determinados na hora do desenvolvimento. A comunicação com os módulos nRF24L01+ é do tipo *handshake*, ou seja, a Central de Controle envia o dado em um duto de transmissão e espera uma resposta no duto de recepção, essa resposta é a garantia de que o dado foi transmitido com sucesso e, além disso, essa resposta contém a informação dos sensores do Periférico que recebeu o dado, portanto esse dado é salvo na tabela de dados, atualizando os valores referentes à leitura de sensores. Após enviar os dados de todos os Periféricos instalados e atualizar a tabela com todas as leituras dos sensores, a função *upload* é executada.

A função *upload* tem como finalidade enviar para o servidor da Aplicação Web a tabela de dados atualizada com os valores atuais dos sensores de cada Periférico. Utilizando

o método POST, cada linha da tabela é enviada ao servidor através de uma requisição HTTP. Os parâmetros utilizados para informar a Aplicação Web de quais dados se tratam são os ID's tanto da Central de Controle quanto do Periférico, esses IDs são únicos e identificam pra Aplicação Web de qual Periférico em qual Central de Controle está sendo atualizado.

3.2.5 Periféricos

Os dispositivos, referenciados nesse trabalho de conclusão de curso como Periféricos, são sistemas embarcados que adicionam funcionalidades físicas ao projeto, esses dispositivos podem ser compostos por vários componentes de *hardware*, como pode ser visualizado na Figura 10. São esses dispositivos que agem diretamente no mundo real, seja com o propósito de monitoramento, como é o caso de sensores, ou controle através de relés.

Os Periféricos consistem de, no mínimo, um microcontrolador e um *driver* externo de comunicação *wireless*. A capacidade de processamento dos periféricos podem ser extremamente reduzidas, dependendo da complexidade do protocolo de comunicação do *driver* externo utilizado.

O objetivo é que esses *drivers* tenham um protocolo simples porém robusto o suficiente para ser utilizada no projeto.

As possibilidades de Periféricos que podem ser desenvolvidos para serem controlados remotamente, com a arquitetura criada nesse trabalho, são inúmeras. Até o presente momento foi desenvolvido um Periférico chamado de Interruptor Inteligente.

São muitas as possibilidades para o desenvolvimento de Periféricos, porém, como o protótipo se limita a demonstrar os conceitos e funcionamento da arquitetura do sistema como um todo, o desenvolvimento ficou limitado a somente um tipo de periférico, explicado a seguir.

3.2.5.1 Interruptor Inteligente

O interruptor inteligente é um exemplo de um possível Periférico, desenvolvido utilizando um Arduino Uno e um módulo de comunicação wireless nRF24L01, o objetivo desse periférico é o de substituir um dos interruptores em um cômodo onde se deseja controlar a iluminação e monitorar o estado das lâmpadas. Também conta com um sensor de movimento que identifica a presença de pessoas no ambiente, sensores estes que podem vir a integrar parte de um sistema de segurança.

Os módulos nRF24L01+ permitem a comunicação dos periféricos com a Central de Controle, a iniciativa da comunicação é sempre da Central de Controle, os periféricos permanecem constantemente monitorando os sensores, porém esses valores só são enviados para a Central de Controle, que por sua vez envia os dados para a Aplicação Web, quando

a Central de Controle toma a iniciativa da conversa e estabelece uma comunicação com o Periférico.

Antes dos detalhes de implementação desse Periférico, é importante uma explicação sobre as principais bibliotecas utilizadas, que também foram usadas no desenvolvimento da Central de Controle, por contar com o mesmo dispositivo RF.

3.2.5.2 Programa

Para a programação do Arduino Uno do Periférico Interruptor Inteligente, foram utilizadas as biblioteca SPI e RF24, explicadas anteriormente.

Um fluxograma com um alto nível de abstração pode ser observado na Figura 16.

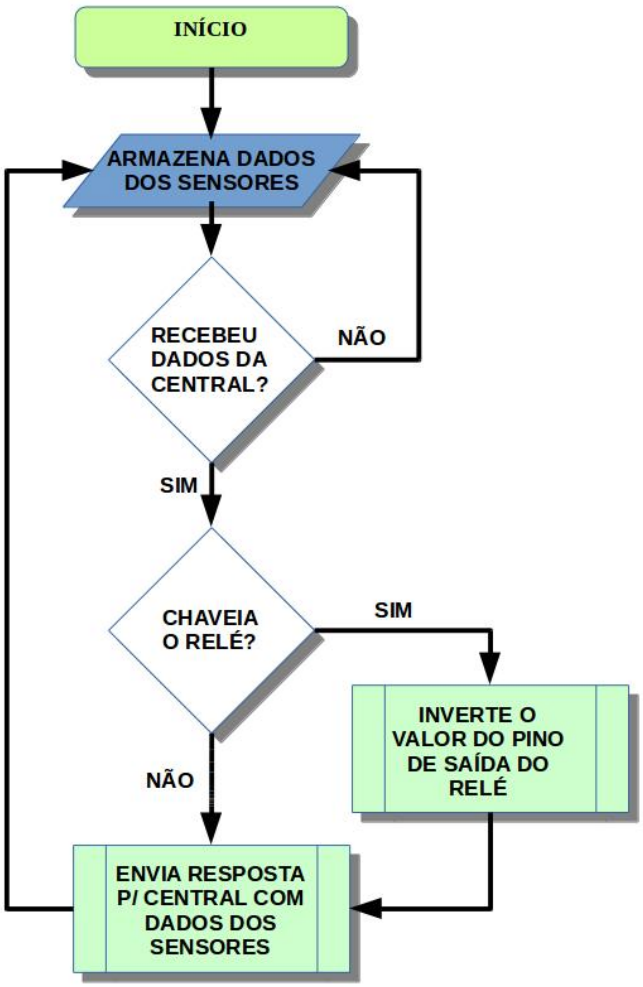

Figura 16: Fluxograma do Periférico Interruptor Inteligente.

Primeiramente as variáveis globais do Periférico e algumas de suas configurações são definidas. Um objeto pertencente à classe RF24 é instanciado, passando como argumento os parâmetros de quais pinos do Arduino Uno estão conectados às portas CE e CS, os dutos de comunicação de leitura e escrita para os módulos nRF24L01 também são definidos.

São criadas as variáveis que irão armazenar os dados recebidos da Central de Controle, e o ID do periférico também é definido, esse é o valor que precisa ser cadastrado pelo usuário na Aplicação Web para que consiga controlar esse dispositivo remotamente, as portas do Arduino Uno que serão utilizadas para leitura dos sensores e controle do relé são definidas.

Na estrutura *setup* do código, os pinos do Arduino Uno referentes aos sensores são definidos como entrada de dados enquanto que o pino referente ao relé é definido como saída de dados. O método *begin* da classe RF24 é então executado, parâmetros do número de tentativas de retransmissão dos dados antes de o pacote ser dado como perdido e o tamanho dos pacotes, em bits, enviados também são definidos, nesse ponto o método *startListening* da classe RF24 é executado, fazendo com que o dispositivo fique monitorando a recepção de dados, à espera de uma mensagem da Central de Controle.

A estrutura *loop* do código, primeiramente atualiza os dados com os valores atuais de leitura dos sensores, feito isso, verifica se existe algum dado para ser recebido no *buffer* do módulo nRF24L01, caso negativo, o processo é repetido. Uma vez que existir algum dado para ser recebido, o Periférico recebe o dado, salva em uma variável e compara seu ID com o contido no dado enviado pela Central de Controle, caso sejam diferentes, ele ignora o dado e retoma o processo de manter as leituras dos sensores atualizadas, caso sejam iguais, ele compara o dado referente ao controle do relé e decide se faz o chaveamento ou não, feito isso, ele envia um pacote de dados de volta à Central de Controle, confirmando que a comunicação foi feita com sucesso e informando o estado atual dos sensores que possui.

3.2.6 Aplicação Móvel

A Aplicação Móvel é um aplicativo para celulares *smartphones*. O requisito mínimo é que os dispositivos precisam de acesso à internet uma vez que o aplicativo não se conecta diretamente à Central de Controle, tanto os comandos de controle quanto os comandos para monitoramento dos Periféricos instalados no sistema são realizados através da API da Aplicação Web.

O aplicativo precisa contar com as seguintes funcionalidades:

1. *Login*: O cadastro de usuários será feito através da Aplicação Web, porém, quando o mesmo deseja utilizar seu *smartphone* para controlar os Periféricos, existe a necessidade do usuário identificar-se perante ao sistema, isso é realizado através de um *login* no aparelho.

2. Listar Centrais de Controle: Uma vez feito o *login*, utilizando a API da Aplicação Web, os dados referentes às Centrais de Controle que o usuário logado controla será recebido. O aplicativo deve então ser capaz de criar uma lista com esses dados recebidos exibindo o nome de cada Central de Controle e, cada item dessa lista,

deve ser um botão que, ao clicar, direciona para uma tela de exibição de Periféricos instalados naquela Central de Controle.

3. Exibir Periféricos: Deve ser uma tela similar a tela de Centrais de Controle, porém, aqui serão listados os Periféricos cadastrados em uma Central de Controle específica. Cada item aqui também deve ser um botão que, ao clicar, direciona para uma tela de exibição daquele Periférico específico.

4. Controlar Periférico: Tela específica de cada Periférico, aqui será exibido as leituras de sensores, caso existam, e um botão que, ao ser clicado, controla o Periférico em questão.

3.2.6.1 Desenvolvimento

A Aplicação Móvel consiste de um aplicativo para *smartphones* com o Sistema Operacional Android. O desenvolvimento do aplicativo do protótipo foi realizado utilizando a *Integrated Development Environment* (IDE) e *Software Development Kit* (SDK) oficiais para desenvolvimento de aplicativos Android, chamada de Android Studio.

A Aplicação Móvel foi desenvolvida em JAVA, uma linguagem de programação orientada a objetos, concorrente, baseada em classes e desenvolvida para ter o mínimo possível de dependências na implementação. Ela foi adotada como linguagem base para os aplicativos construídos para a plataforma Android, sendo assim, como o desenvolvimento foi feito especificamente para essa plataforma e utilizada suas ferramentas, essa foi a linguagem de programação adotada.

Para explicar o funcionamento, esse trabalho resume o desenvolvimento da Aplicação Móvel, que nesse projeto é um aplicativo Android, limitando-se à explicar as *activitys* existentes. São elas:

1. *Login*: Essa é a *activity* relacionada à tela principal do aplicativo, o aplicativo é iniciado através dessa atividade. A tela consiste de dois campos para o usuário inserir o email e a senha do seu cadastro no sistema. Uma vez confirmado o cadastro do usuário pelo servidor, a informação do ID do usuário é salva e passada para a próxima *activity* chamada *Devices*.

2. *Devices*: Com a informação do usuário, uma requisição é feita ao servidor sobre os dispositivos, nesse trabalho tratados como Centrais de Controle, que o mesmo tem acesso. Uma lista é gerada e possibilita que o mesmo escolha entre as Centrais de Controle, uma vez que uma é clicada, essas informações são salvas e passadas para a próxima *activity*, chamada *Peripherals*.

3. *Peripherals*: Essa é a tela responsável por mostrar ao usuário todos os Periféricos que existem instalados na Central de Controle selecionada, uma lista desses periféricos é gerada através dos dados fornecidos pelo servidor. Uma vez escolhido

um Periférico que o usuário deseja interagir, essas informações são passadas para a próxima *activity*, chamada de *Sensors*.

4. *Sensors*: Aqui são mostrados ao usuário todos os componentes e funcionalidades existentes no Periférico da Central de Controle que o mesmo deseja interagir, no caso do interruptor inteligente, desenvolvido nesse protótipo, ele conta com dois sensores, que seus estados são mostrados na tela e um botão para que o usuário possa controlar remotamente esse periférico.

Além das *activitys* e os *layouts* associados às mesmas, foram necessárias duas classes auxiliares, uma responsável por tratar as requisições HTTP realizadas ao servidor, e outra utilizada para tratar os dados recebidos e gerar as listas de forma dinâmica.

3.2.7 Relacionamento dos componentes com o Modelo de Domínio

Nessa seção são apresentados os resultados do desenvolvimento do protótipo e como esses componentes se relacionam com o modelo desenvolvido através do projeto IoT-A.

Todas as telas da Aplicação Web que são mostradas a seguir, estão associadas com os serviços de *frontend* descritos no Modelo de Domínio (Figura 10), esses serviços são expostos através dessas telas. A Figura 18 é a tela onde um novo usuário pode ser cadastrar, no Modelo de Domínio essa tela é representada pelo componente de *software* *Login*, da Aplicação Web.

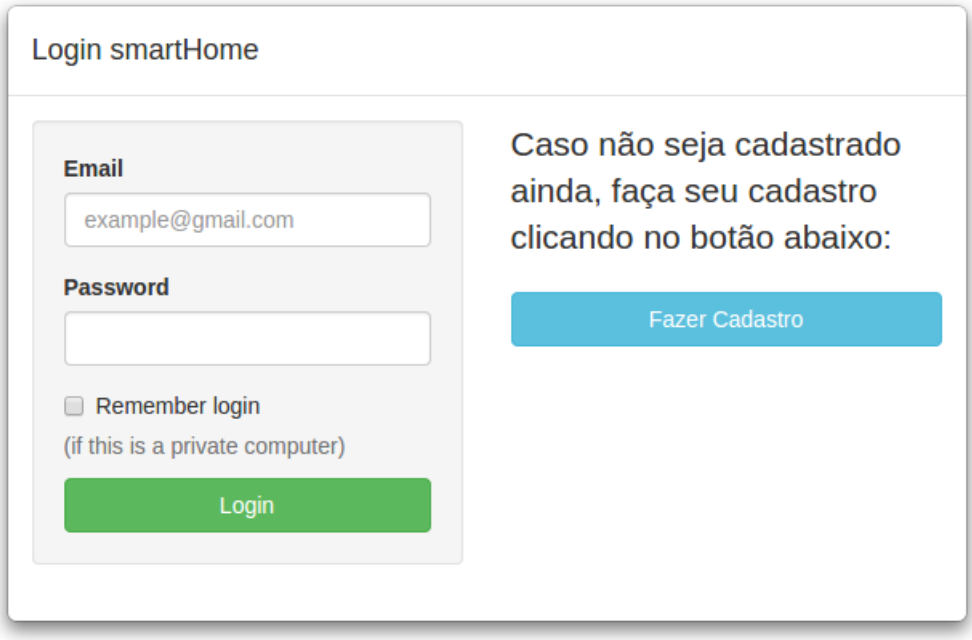

Figura 17: Tela de login da Aplicação Web.

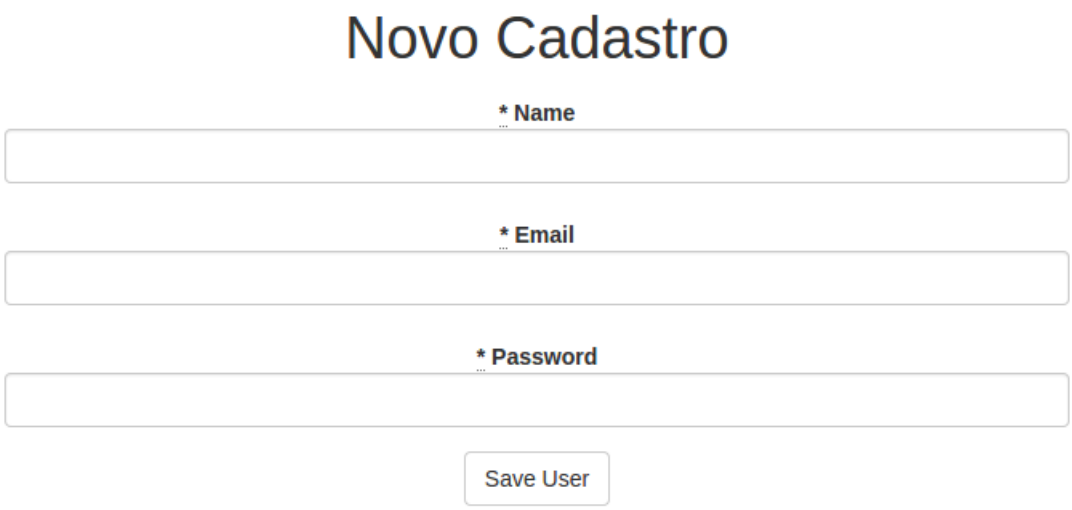

Figura 18: Tela de cadastro da Aplicação Web.

A partir do momento que o usuário se identifica na Aplicação Web uma tela exibe para o mesmo as Centrais de Controle que ele tem acesso (Figura 19) e permite que uma nova Central de Controle seja criada no sistema. Essas telas de configuração estão associadas com o componente de *software* chamado de *Preset Configuration*, como descrito no modelo.

Uma vez escolhida a Central que deseja acessar, ele pode verificar as pessoas que podem monitorar e controlar essa Central de Controle (Figura 20), ou pode navegar para os Periféricos que estão cadastrados na mesma, os Periféricos disponíveis nessa Central de Controle e permite que o usuário edite, exclua ou crie um novo Periférico associado à essa Central (Figura 21). Também é possível criar um novo Periférico, inserindo o nome que deseja, o PeripheralID fornecido pelo fabricante e o tipo do Periférico (Figura 22). No Modelo de Domínio, essas telas estão descritas tanto pelo *Preset Configuration* quanto pelos componentes chamados de Leitura de Sensores e Controle Remoto. Todos esses serviços que são expostos por essas telas, invocam recursos do banco de dados do sistema através de consultas ao banco de dados.

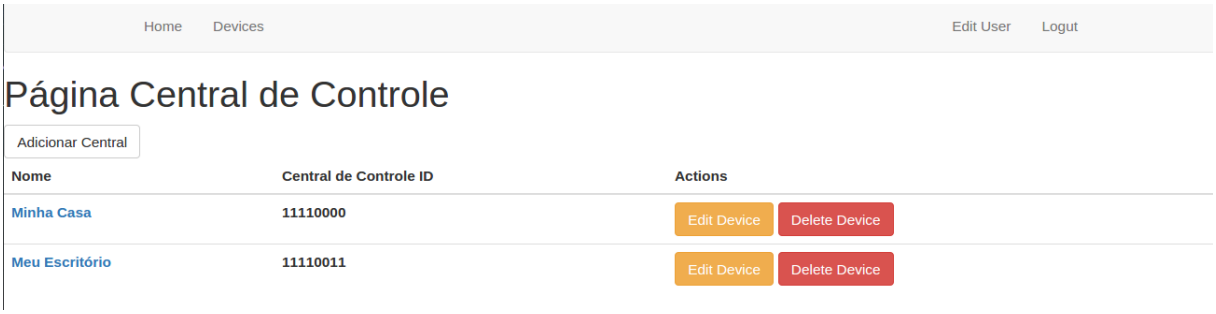

Figura 19: Tela de Centrais de Controle da Aplicação Web.

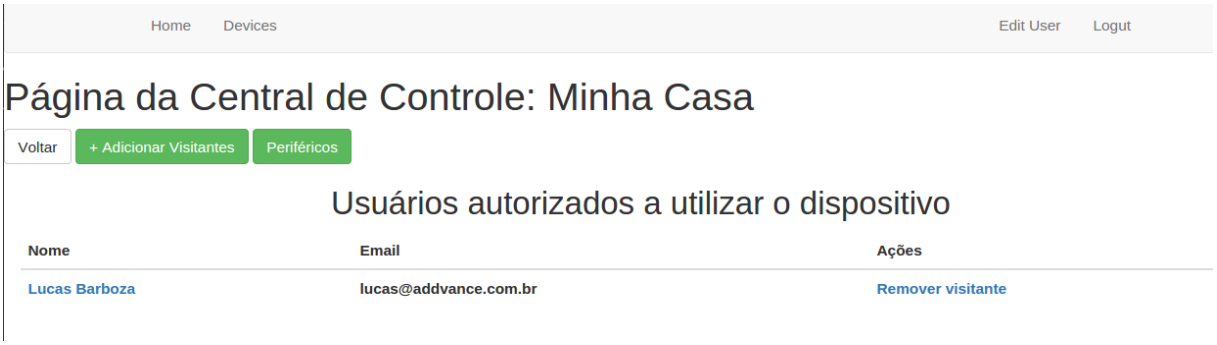

Figura 20: Tela da Central de Controle da Aplicação Web.

Figura 21: Tela de Periféricos da Aplicação Web.

Após fazer a configuração do sistema através da Aplicação Web, o usuário pode utilizar seu *smartphone* e fazer o *login* (Figura 23). Todas as telas e componentes de *software* associados ao *smartphone* estão descritos no modelo pelos componentes chamados de Aplicação Móvel, onde estão representados os serviços de *Login*, Leitura do Sensor e Controle Remoto. Também existe uma tela onde o mesmo pode verificar as Centrais de Controle que tem acesso (Figura 24) e, ao escolher alguma, é redirecionado para a tela que exibe os Periféricos cadastrados nessa Central de Controle (Figura 25). Ao selecionar o Periférico desejado, o usuário pode monitorar os estados dos sensores presentes no mesmo e controlar o Periférico (Figura 26). Esses recursos oferecidos pelo *smartphone* também invocam recursos da Aplicação Web através de requisições HTTP ao servidor, onde o mesmo responde com os dados solicitados através de acessos ao banco de dados.

Figura 22: Tela de Novo Periféricos Aplicação Web.

O protótipo foi então montado em uma maquete para demonstração dos conceitos, a parte dos fundos onde estão instalados os componentes pode ser vista na Figura 27, onde é possível visualizar instalados 3 Periféricos, presos na parte de trás da maquete e uma Central de Controle.

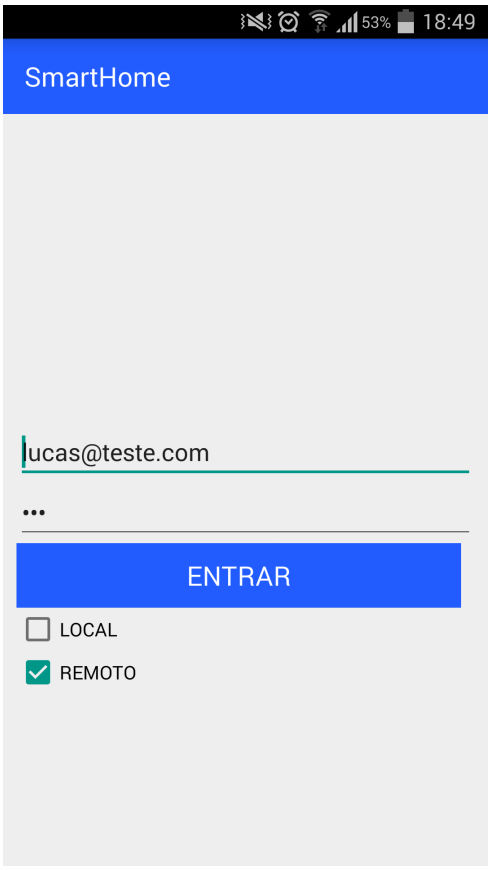

Figura 23: Tela de login da Aplicação Móvel.

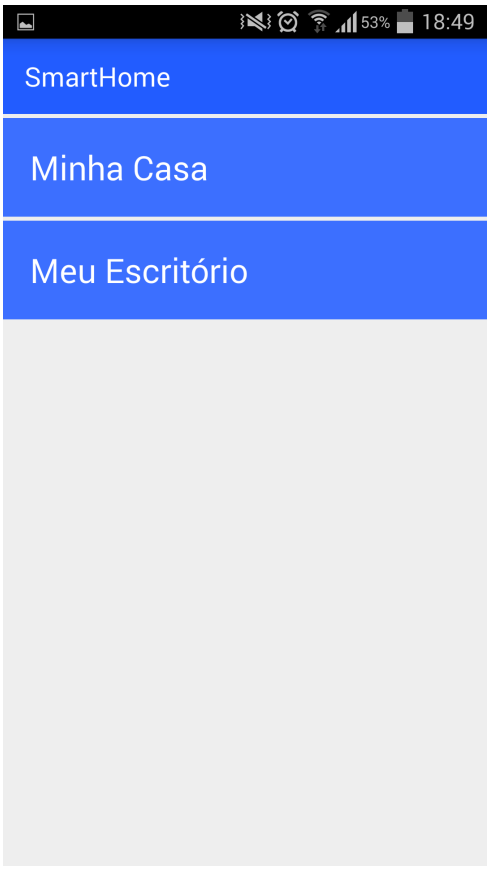

Figura 24: Tela de Centrais de Controle da Aplicação Móvel.

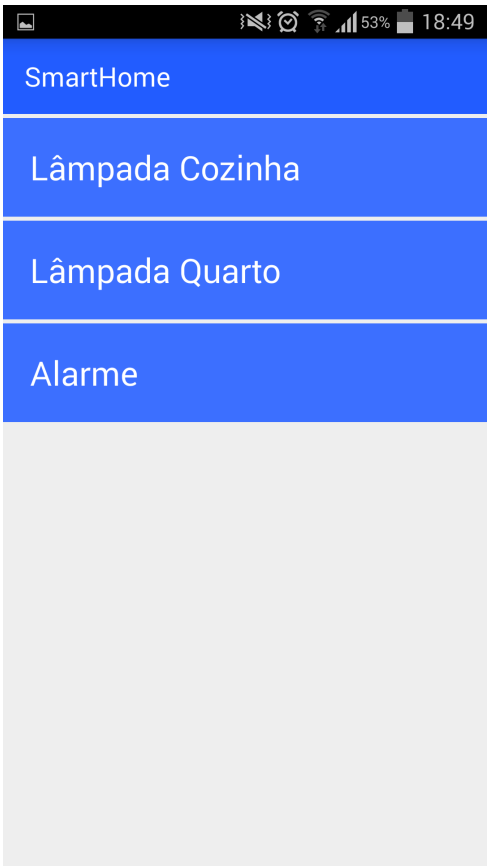

Figura 25: Tela de Periféricos da Aplicação Móvel.

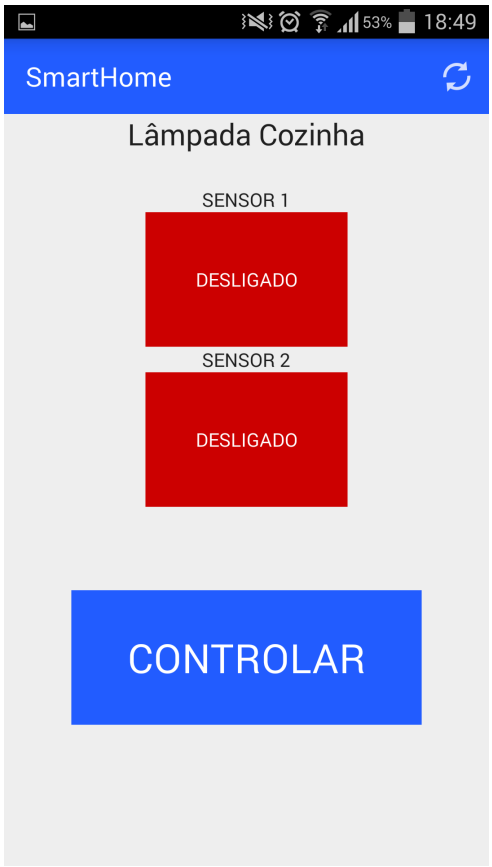

Figura 26: Tela do Periférico da Aplicação Móvel.

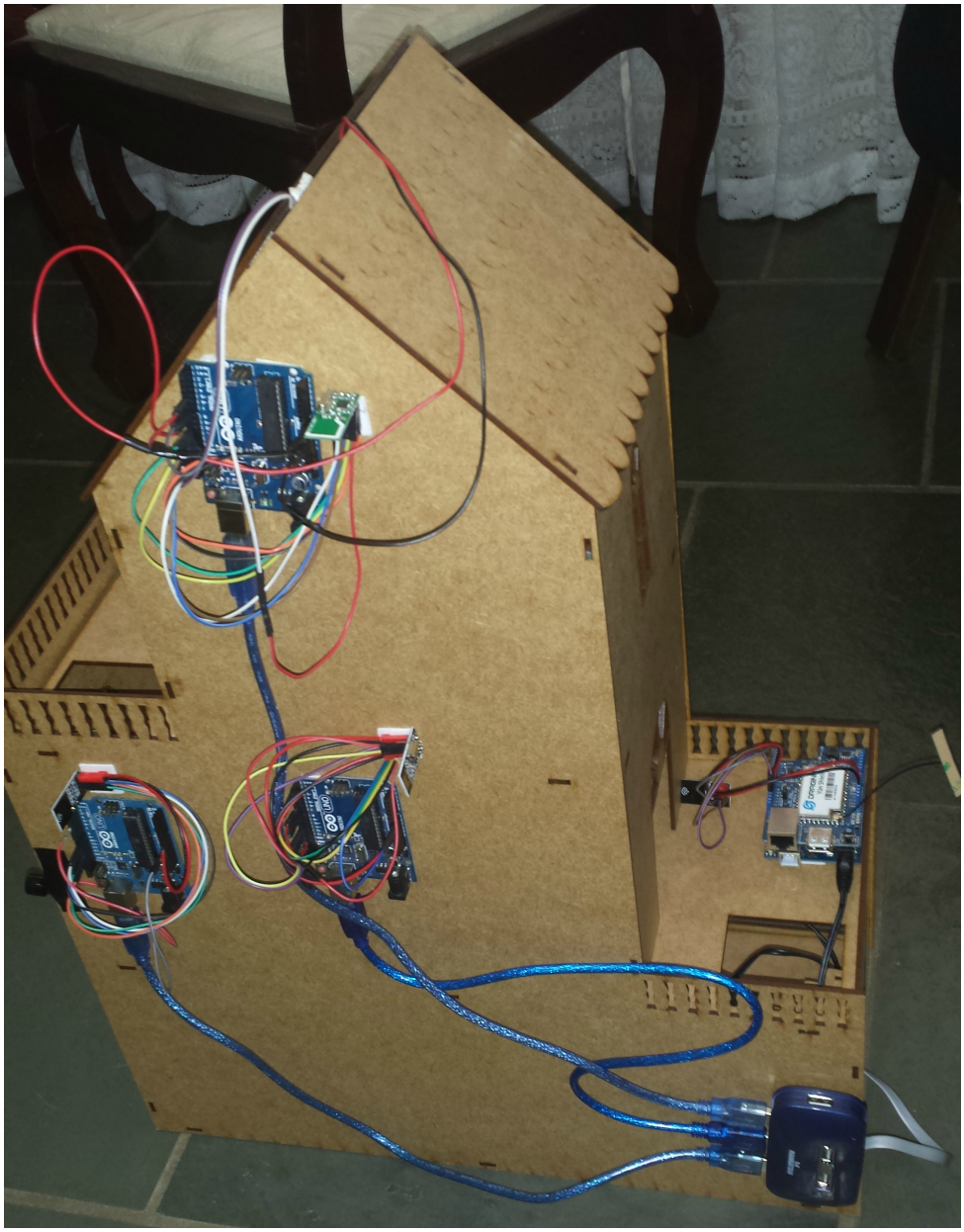

Figura 27: Maquete com o protótipo.

Capítulo 4

Resultados

Como resultados desse trabalho, é possível visualizar os dados presentes no banco de dados da Aplicação Web. Os cadastros são referentes ao protótipo apresentado. A Figura 28, Figura 29 e Figura 30 representam, respectivamente, as entradas no banco de dados referentes aos Usuários, Centrais de Controle e Periféricos.

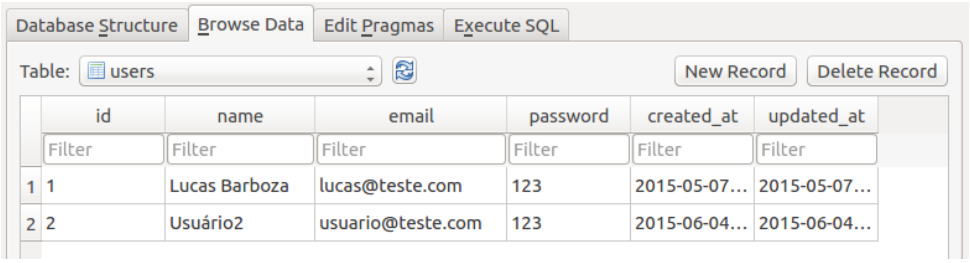

Figura 28: Tabela de usuários.

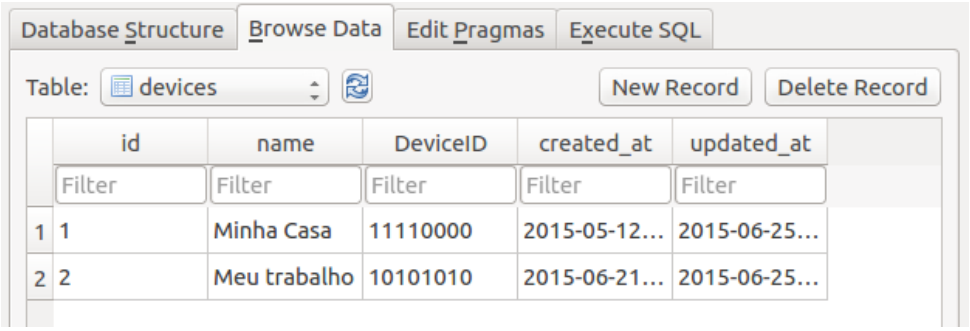

Figura 29: Tabela de Centrais de Controle.

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: peripherals — New Record — Delete Record

	id	name	control	device_id	peripheralID	sensor1	sensor2
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	13	Quarto Superior	f	7	1101	f	t
2	20	Quarto meio	t	7	1001	t	t
3	21	Alarme	f	7	1000	f	f

Figura 30: Tabela de Periféricos.

Os recursos de *download* e *upload* que estão associados à Central de Controle descritos no Modelo de Domínio, são requisições ao servidor que, através de acesso ao seu banco de dados, enviam os dados requiridos pela Central ou atualizam os dados enviados pela Central, esses recursos são mostrados através da resposta do servido representados, respectivamente, através da Figura 31 e Figura 32.

```
Started GET "/getCentralInfo/11110000" for 127.0.0.1 at 2015-06-28 10:03:08 -0300
Started GET "/getCentralInfo/11110000" for 127.0.0.1 at 2015-06-28 10:03:08 -0300
Processing by TasksController#getInfo as HTML
Processing by TasksController#getInfo as HTML
  Parameters: {"device_id"=>"11110000"}
  Parameters: {"device_id"=>"11110000"}
  Device Load (0.6ms)  SELECT  "devices".* FROM "devices" WHERE "devices"."DeviceID" = ? LIMIT 1  [["DeviceID", "11110000"]]
  Device Load (0.6ms)  SELECT  "devices".* FROM "devices" WHERE "devices"."DeviceID" = ? LIMIT 1  [["DeviceID", "11110000"]]
  Peripheral Load (0.3ms)  SELECT "peripherals".* FROM "peripherals" WHERE "peripherals"."device_id" = ?  [["device_id", "2"]]
  Peripheral Load (0.3ms)  SELECT "peripherals".* FROM "peripherals" WHERE "peripherals"."device_id" = ?  [["device_id", "2"]]
  Rendered tasks/getInfo.html.erb within layouts/task (1.6ms)
  Rendered tasks/getInfo.html.erb within layouts/task (1.6ms)
Completed 200 OK in 38ms (Views: 7.7ms | ActiveRecord: 1.4ms)
Completed 200 OK in 38ms (Views: 7.7ms | ActiveRecord: 1.4ms)
```

Figura 31: Download dos dados da Central de Controle.

```
Started GET "/updateSensor/11110000/1001/2/0" for 127.0.0.1 at 2015-06-28 10:04:12 -0300
Started GET "/updateSensor/11110000/1001/2/0" for 127.0.0.1 at 2015-06-28 10:04:12 -0300
Processing by TasksController#updateSensor as HTML
Processing by TasksController#updateSensor as HTML
  Parameters: {"device_id"=>"11110000", "peripheral_id"=>"1001", "sensor"=>"2", "value"=>"0"}
  Parameters: {"device_id"=>"11110000", "peripheral_id"=>"1001", "sensor"=>"2", "value"=>"0"}
  Device Load (0.1ms)  SELECT  "devices".* FROM "devices" WHERE "devices"."DeviceID" = ? LIMIT 1  [["DeviceID", "11110000"]]
  Device Load (0.1ms)  SELECT  "devices".* FROM "devices" WHERE "devices"."DeviceID" = ? LIMIT 1  [["DeviceID", "11110000"]]
  Peripheral Load (0.3ms)  SELECT  "peripherals".* FROM "peripherals" WHERE "peripherals"."peripheralID" = ? AND "peripherals"."device_id" = ? L
IMIT 1  [["peripheralID", "1001"], ["device_id", "2"]]
  Peripheral Load (0.3ms)  SELECT  "peripherals".* FROM "peripherals" WHERE "peripherals"."peripheralID" = ? AND "peripherals"."device_id" = ? L
IMIT 1  [["peripheralID", "1001"], ["device_id", "2"]]
   (0.1ms)  begin transaction
   (0.1ms)  begin transaction
   (0.1ms)  commit transaction
   (0.1ms)  commit transaction
  Rendered tasks/updateSensor.html.erb within layouts/task (0.4ms)
  Rendered tasks/updateSensor.html.erb within layouts/task (0.4ms)
Completed 200 OK in 12ms (Views: 2.6ms | ActiveRecord: 0.7ms)
Completed 200 OK in 12ms (Views: 2.6ms | ActiveRecord: 0.7ms)
```

Figura 32: Upload dos dados da Central de Controle.

Os recursos da Aplicação Móvel para visualizar os estados dos sensores e controlar um Periférico podem ser visualizados através da reposta do servidor às requisições do *smartphone* na Figura 33 e Figura 34, respectivamente. Após a requisição enviada pelo *smartphone* de controlar o Periférico, é possível visualizar que o mesmo foi controlado através da Figura 35. Também é possível ver as telas do *smartphone* indicando que a lâmpada do cômodo onde está instalado o Periférico esta acesa na Figura 36 e quando a luz está acesa e o sensor de movimento está detectando alguma agitação no cômodo (Figura 37).

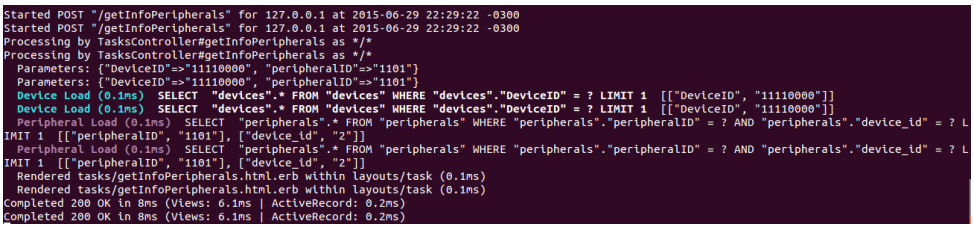

Figura 33: Requisição dos estados dos sensores da Aplicação Móvel.

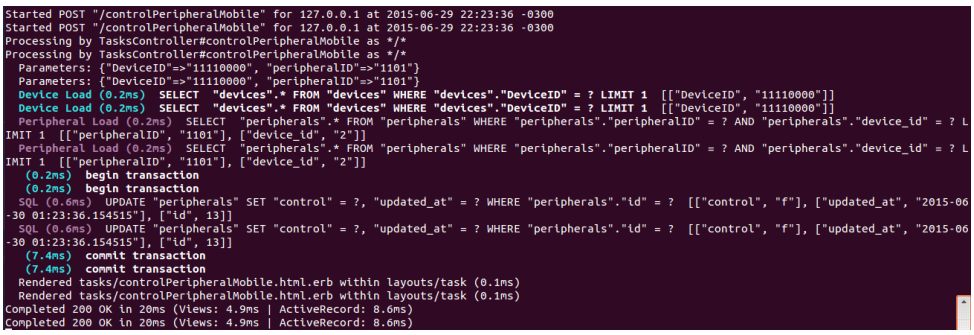

Figura 34: Controle de um Periférico pela Aplicação Móvel.

Figura 35: Periférico Controlado pela Aplicação Móvel.

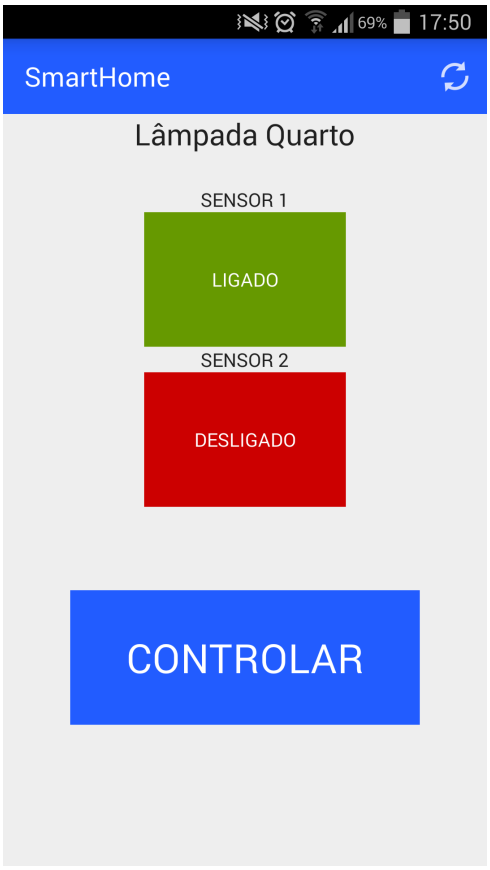

Figura 36: Tela de um Periférico leitura dos sensores.

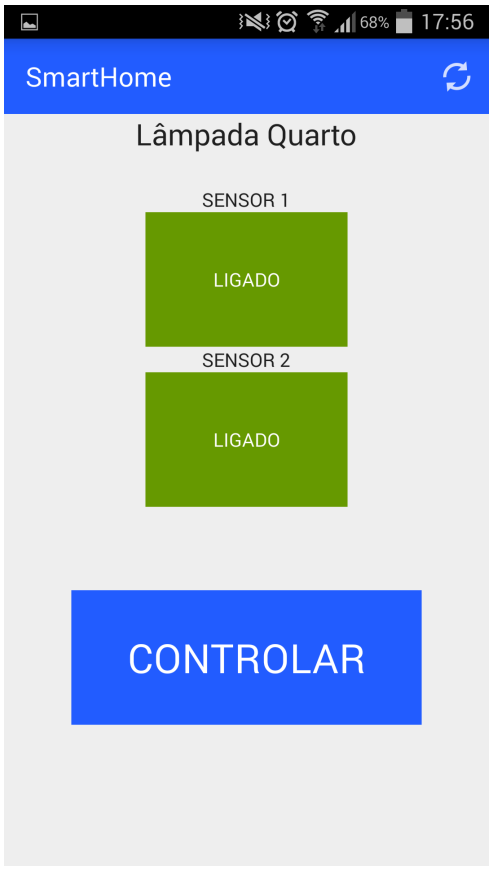

Figura 37: Tela de um Periférico leitura dos sensores.

Capítulo 5

Discussões e Conclusão

Nesse capítulo são discutidos os resultados encontrados, as dificuldades durante o processo de desenvolvimento e algumas discussões relevantes sobre as tecnologias e produtos utilizados no protótipo.

5.1 Discussões

Os resultados obtidos do desenvolvimento deste projeto foram promissores, a arquitetura do sistema resultante é escalável e aproxima-se bastante de uma possível estrutura para um *Minimum Viable Product* (MVP) de um produto comercializável. Esse fato comprova a viabilidade da utilização das modelagens propostas pelo projeto IoT-A. A seguir serão apresentados comentários e discussões referentes a cada componente do sistema desenvolvido.

5.1.1 Aplicação Web

A Aplicação Web é, sem dúvida, o grande gargalo do sistema quando o assunto é escalabilidade. Uma vez que é através da sua API, que está hospedada no mesmo servidor que também trata as navegações nos *browsers* dos usuários do sistema, que todo o funcionamento ocorre. Requisições HTTP da Aplicação Móvel e Central de Controle são constantes, toda informação precisa passar pela Aplicação Web, uma vez que é lá onde as informações são armazenadas nos bancos de dados.

O motivo pelo qual foi decidido no *design* do sistema por não hospedar a Aplicação Web na própria Central de Controle, o que resolveria o problema de escalabilidade pois, cada sistema, teria seu próprio servidor, diferentemente do modelo atual onde tudo fica concentrado em um único servidor na nuvem, foi o fato de não ser uma tarefa fácil tornar esse servidor local visível na internet. Para um usuário comum, alvo do desenvolvimento desse projeto, alterar configurações no roteador de sua rede local, encontrar seu IP de internet, que na maioria dos casos desses usuários é um IP dinâmico, IP local da Cen-

tral de Controle nessa rede local e juntar todas essas e outras informações para fazer a configuração do sistema não seria uma tarefa trivial. O servidor web centralizado resolve todos esses problemas, tirando a responsabilidade dessas configurações do usuário final.

A velocidade do servidor em responder uma requisição e a forma como o mesmo trata as concorrências são fatores extremamente críticos nesse projeto. Uma vez que a Central de Controle está constantemente requisitando os dados do banco de dados do servidor, um *delay* de resposta do servidor a uma requisição da Central de Controle impacta diretamente na experiência do usuário em, ao apertar um botão de controle, o comando demorar a acontecer.

Uma vez que todo o servidor é um *Cloud Service*, a responsabilidade de conseguir manter e responder todas as mensagens com um desempenho satisfatório, ou seja, para que o usuário final não perceba nenhum *delay* em suas ações, é toda do serviço contratado. Caso viessem a existir milhares de Centrais de Controle, em um cenário onde o sistema estaria instalado em milhares de lugares diferentes, um redimensionamento da capacidade do servidor seria necessário. Apesar de crítico, esse problema é facilmente solucionável e precisa ser tratado a medida que a adesão ao sistema cresça.

Com o auxílio do *framework* Ruby-on-Rails, foi possível criar uma interface gráfica para o usuário bem intuitiva e fácil de configurar. Um dos objetivos, que o sistema fosse de fácil configuração, foi totalmente alcançado na Aplicação Web. Um usuário que viesse a instalar o sistema em sua residência, conseguiria através do site desenvolvido, criar uma conta, cadastrar os IDs dos dispositivos colocados em sua residência e, automaticamente, esse usuário passaria a controlar e monitorar remotamente os Periféricos.

5.1.2 Central de Controle

A Central de Controle mostrou-se estável e robusta, permaneceu durante semanas com o sistema ligado e não apresentou nenhum problema que o sistema tenha "quebrado" e precisasse ser reiniciado. Com a utilização do *Shield* Dragino Yún, que oferece grande capacidade de processamento de dados, o tempo que o sistema demora para processar os dados, uma vez que o *download* dos mesmos é feito, é zero, isso também se deve ao fato de os dados,fornecidos pelo servidor, serem otimizados para essa aplicação.

Uma vez que a Central de Controle possui um funcionamento sequencial e a mesma possui um *timeout* de 100ms para uma resposta do Periférico, é possível concluir, a partir de uma análise do fluxograma da mesma, que conforme a quantidade de pacotes perdidos aumenta, o tempo de resposta do sistema aumenta proporcionalmente. Por exemplo, quando 10 pacotes são perdidos um *delay* de, no mínimo, 1 segundo é adicionado ao tempo de resposta, o que causa uma sensação ruim na experiência do usuário, que espera uma resposta instantânea quando controla seus Periféricos.

Um problema que também impacta em uma experiência ruim do usuário do sistema é um erro que o mesmo possa vir a cometer na hora de configurar seus Periféricos, caso ele

possua periféricos cadastrados na Aplicação Web que não estão fisicamente instalados no mesmo cenário de sua Central de Controle, ou que estejam com o ID errado ou fora do range de alcance da Central, cada Periférico que esteja cadastrado errado vai adicionar um *delay* no tempo de resposta de cada ciclo, impactando assim negativamente na sua experiência utilizando o sistema.

5.1.3 Periféricos

O tempo de resposta do Periférico desenvolvido, batizado de Interruptor Inteligente, foi excelente, ou seja, uma vez recebido o dado da Central de Controle, se esse contiver a informação para que o mesmo chaveie, o tempo que o dispositivo demora para interpretar esse dado, chavear e responder para a Central de Controle com os valores lidos dos sensores é zero, uma vez que os dados foram desenvolvidos para minimizar a quantidade de processamento necessária. A Central de Controle possui um *timeout* de 100ms, portanto é razoável supor que o tempo que o Periférico demora em sua rotina deve ser bem menor que esse valor.

5.1.4 Aplicação Móvel

Apesar de se tratar de uma aplicação leve, relativamente simples e com poucas funcionalidades, a dificuldade no desenvolvimento foi grande, devido a falta de conhecimento prévio da linguagem Java e da estrutura dos aplicativos Android. O aplicativo foi desenvolvido focado na facilidade de utilização, minimizando as informações na tela e tornando-o assim intuitivo para os usuários.

O Aplicativo Móvel, assim como a Central de Controle, depende totalmente da internet, seu tempo de resposta é similar ao da Central de Controle na requisição dos dados do servidor. Uma vez que poucos dados são processados no aparelho e esses são enviados da Aplicação Web de uma forma que sejam fáceis de tratar no aplicativo, praticamente todo o tempo que o mesmo demora na navegação entre as telas é devido às requisições ao servidor, e essa demora depende totalmente da velocidade de conexão do *smartphone*.

5.2 Conclusão

O objetivo de utilizar uma arquitetura de referência que guia desenvolvedores de sistemas na criação de soluções de Internet das Coisas foi alcançado com sucesso, o projeto IoT-A auxiliou no desenvolvimento para que aspectos importantes de sistemas IoT fossem tratados, aspectos como escalabilidade discutidos nesse trabalho.

Os objetivos propostos no escopo do projeto foram alcançados, a arquitetura do sistema provou-se funcional, escalável e de fácil configuração através do protótipo desenvolvido. Na fase de prototipagem, os quatro componentes propostos, referenciados durante

o trabalho como Central de Controle, Periféricos, Aplicação Web e Aplicação Móvel foram desenvolvidos com sucesso e provaram a estabilidade do sistema e, portanto, da arquitetura que gerou esse modelo. Esse trabalho mostrou que a arquitetura do sistema desenvolvido pode ser utilizada para um possível produto comercial com aplicações reais, através dos requisitos discutidos é possível derivar um produto mais barato que as soluções presentes hoje no mercado, tornando-o assim competitivo e interessante.

O grande conhecimento adquirido durante o desenvolvimento desse trabalho de conclusão de curso, mostrou que algumas das tecnologias utilizadas no protótipo não eram as ideais para a aplicação. Como proposta para futuros trabalhos fica, principalmente: Melhorar a comunicação entre a Central de Controle e os Periféricos, isso pode ser feito implementando a rede *mesh* nos módulos nRF24L01 utilizados e, consequentemente, aumentar o alcance da rede ou utilizar outras tecnologias disponíveis. Implementar uma comunicação persistente entre a Central de Controle e a Aplicação Web, constantes requisições HTTP sobrecarregam mais o sistema do que conexões do tipo *WebSocket*, por exemplo. Desenvolver novos Periféricos com outras funcionalidades, basicamente qualquer coisa que possa ser controlada pode ser adaptada ao sistema. Aumentar a segurança do sistema, utilizando criptografias nas comunicações, implementar HTTPS no servidor, cifrar os banco de dados e mensagens trocadas entre a Central de Controle e os Periféricos.

Referências Bibliográficas

ARDUINO. **Arduino Shields**. [S.l.], 2014. Disponível em: <http://arduino.cc/en-/Main/ArduinoShields>. Acesso em: 14.04.2015.

_____. **Arduino Uno**. [S.l.], 2014. Disponível em: <http://arduino.cc/en/Main-/ArduinoBoardUno>. Acesso em: 14.04.2015.

_____. **What is Arduino?** [S.l.], 2014. Disponível em: <http://arduino.cc/en/Guide-/Introduction>. Acesso em: 12.04.2015.

_____. **Bridge Library for Arduino Yun**. [S.l.], 2015. Disponível em: <http://www-.arduino.cc/en/Reference/YunBridgeLibrary>. Acesso em: 14.04.2015.

ATHEROS. **AR9331 Highly-Integrated and Cost Effective IEEE 802.11n 1x1 2.4 GHz SoC for AP and Router Platforms**. [S.l.], 2010. Disponível em: <https://www.openhacks.com/uploadsproductos/ar9331_datasheet.pdf>. Acesso em: 02.05.2015.

BAUER, M. et al. **Introduction to the Architectural Reference Model for the Internet of Things**. IoT-A, 2011. Disponível em: <http:///www.iot-a.eu>.

_____. **Final architectural reference model for the IoT**. IoT-A, 2013. Disponível em: <http:///www.iot-a.eu>.

BERNERS-LEE, T. **Hypertext Transfer Protocol – HTTP/1.0**. [S.l.], 1996. Disponível em: <http://tools.ietf.org/html/rfc1945>. Acesso em: 10.05.2015.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. **Hypertext Transfer Protocol – HTTP/1.0, RFC1945**. [S.l.]: Network Working Group, 1996.

BRADLEY, J.; BARBIER, J.; HANDLER, D. **Embracing the Internet of Everything To Capture Your Share of 14.4 Trillion dollars**. [S.l.], 2013. Disponível em: <http://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy-.pdf>. Acesso em: 21.02.2015.

BUTLER, J. et al. **Wireless Networking in the Developing World**. [S.l.: s.n.], 2013.

CAELUM. **Desenvolvimento Web com Ruby on Rails**. [S.l.], 2004. Disponível em: <http://www.caelum.com.br/apostila-ruby-on-rails/>. Acesso em: 14.04.2015.

CLINTON, J. **Ruby Phrasebook**. [S.l.], 2009.

DRAGINO. **Yun Shield**. [S.l.], 2014. Disponível em: <http://www.dragino.com-/products/yunshield/item/86-yun-shield.html>. Acesso em: 14.04.2015.

FAYAD, M. E.; SCHMIDT, D. C. **Object-Oriented Application Frameworks**. [S.l.], 1997.

FOLDOC. **Free On-line Dictionary Of Computing**. [S.l.], 1995. Disponível em: <http://foldoc.org/ApplicationProgramInterface>. Acesso em: 12.05.2015.

FOROUZAN, B. A. **TCP/IP Protocol Suite**. 4rd. ed. New York, NY, USA: McGraw-Hill Companies, 2010.

JUNGLING, M.; WOOD, P. A. The internet of things is now: Connecting the real economy. **Morgan Stanley Blue Papers**, 2014. 2014.

MANYIKA, J. et al. Disruptive technologies: Advances that will transform life, business, and the global economy. **McKinsey Global Institute**, 2013. p. 51–60, 2013.

MEDEIROS, J. C. de O. **Princípiois de Comunicação**. 2nd. ed. [S.l.]: Editora Érica, 2007.

REENSKAUG, T. **The original MVC reports**. [S.l.], 1979.

RIVA, F. de la. **8 ideias de negcóios promissores para 2015**. [S.l.], 2014. Disponível em: <http://exame.abril.com.br/pme/noticias/8-ideias-de-negocios-promissores-para-2015>. Acesso em: 27.03.2015.

SEMICONDUTOR, N. **nRF24L01 Ultra low power 2.4GHz RF Transceiver**. [S.l.], 2014. Disponível em: <http://www.nordicsemi.com/eng/Products/2.4GHz-RF-/nRF24L01P>. Acesso em: 17.04.2015.

SHIBU. **Introduction to Embedded Systems 1E**. [S.l.]: Tata McGraw-Hill Education, 2009.

STEWART, B. **An Interview with the Creator of Ruby**. [S.l.], 2001. Disponível em: <http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html>. Acesso em: 21.04.2015.