Felipe Artur Mariano

Integração de tecnologias da Internet das Coisas a um supervisor de energia através da programação orientada à fluxos

São José - SC

Abril de 2016

Felipe Artur Mariano

Integração de tecnologias da Internet das Coisas a um supervisor de energia através da programação orientada à fluxos

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Instituto Federal de Santa Catarina - IFSC

Curso Superior de Tecnologia em Sistemas de Telecomunicacões

Orientador: Prof. Eraldo Silveira e Silva, Dr. Eng.

São José - SC

Abril de 2016

Felipe Artur Mariano

Integração de tecnologias da Internet das Coisas a um supervisor de energia através da programação orientada à fluxos

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Trabalho aprovado. São José - SC, 1 de abril de 2016:

Prof. Eraldo Silveira e Silva, Dr. Eng.
Orientador

Prof. Pedro Armando da Silva Jr, Dr. Eng.
Membro da banca

Prof. Tiago Semprebom, Dr. Eng.
Membro da banca

São José - SC
Abril de 2016

Agradecimentos

Primeiramente, agradeço a minha família por toda sua paciência, confiança e apoio, os quais permitiram que eu chegasse a este ponto.

Agradeço a todos do campus IFSC-SJ pelas oportunidades de aprendizagem e pelo auxílio fornecidos ao longo do curso.

Agradeço ao meu orientador professor Eraldo Silveira e Silva pelo tempo, conhecimento e atenção prestados, além de todos os outros professores, funcionários e alunos que me ajudaram a realizar este trabalho.

Resumo

Este trabalho teve como objetivo inicial implantar melhorias no sistema supervisório de energia do IFSC-SJ. O servidor EmonCMS, utilizado neste sistema, era capaz de processar medições enviadas por nós remotos e registrá-los em um banco de dados. Porém, ele não provê um mecanismo ativo de resposta à eventos, o que dificulta a criação de uma aplicação para o acionamento remoto de atuadores. Além de que, o protocolo usado nas mensagens entre os nós e o servidor, o *Hyper Text Transfer Protocol* (HTTP), não é adequado para este propósito. Ele é desnecessariamente complexo para dispositivos com recursos limitados como o Arduíno e não oferece uma função *multicast* caso o numero de nós do sistema seja expandido.

Neste trabalho propõe-se um esquema de integração do sistema legado com a plataforma Node-RED, resolvendo parte das deficiências do servidor EmonCMS. O servidor Node-RED permite a criação de aplicações com programação orientada a fluxo e possui nós para uso do protocolo *Message Queue Telemetry Transport* (MQTT) ambos voltados para a Internet das Coisas. Uma aplicação para prova de conceito foi criada sobre este sistema demonstrando a potencialidade do mesmo e garantindo o funcionamento do sistema anterior.

O servidor Node-RED mostrou ser capaz de facilitar a criação e integração de aplicações diversas, enquanto o protocolo MQTT mostrou ser um substituto adequado ao HTTP.

Palavras-chaves: Sistema de Monitoramento de Energia, Arduíno, Aplicação Web, MQTT, Node-RED, Programação orientada à fluxo.

Abstract

This work enhanced a energy supervisor installed in the IFSC-SJ campus. The EmonCMS server, used by the supervision system previously, was capable of processing measurements sent by remote nodes and registering them in a database. However, it did not provide an active event response mechanism, which made difficult the creation of an aplication for remote actuation. In addition, the HTTP protocol used in the messages exchanged between the nodes and the server, showed to be inaproprieted for this purpose, as it is unnecessarily complex for resource constrained devices like the Arduino and it does not offer a multicast function if the number of nodes in the system is expanded.

This work addressed those deficiencies through the use of the Node-RED platform, which permits the rapid creation of aplications based on flow-orientted programing interface, and the MQTT protocol, created for Internet of Things aplicattions. An integration model with the EmonCMS server was then proposed and a proof of concept aplicattions were presented.

The Node-RED server proved to be capable of easing the creation and integration of applications, while the MQTT protocol proved to be an adequate substitute to the HTTP protocol.

Key-words: Energy Monitoring System , Arduino, Web application, MQTT, Node-RED, Flow-oriented programming.

Lista de ilustrações

Lista de tabelas

Listagens

Lista de Abreviaturas

AC *Alternating Current*

AJAX *Asynchronous Javascript and XML*

API *Application Programming Interface*

CSS *Cascading Style Sheets*

CSV *Comma-separated values*

DHCP *Dynamic Host Configuration Protocol*

HTML *Hyper Text Markup Language*

HTTP *Hyper Text Transfer Protocol*

IDE *Integrated Drive Electronics*

IoT *Internet of Things*

JSON *JavaScript Object Notation*

LCD *Liquid Crystal Display*

MAC *Media Access Control*

MIME *Multipurpose Internet Mail Extensions*

MQTT *Message Queue Telemetry Transport*

MVC *Model View Controller*

PHP *PHP: Hyper Text Preprocessor*

PoE *Power over Ethernet*

QoS *Quality of Service*

SQL *Structured Query Language*

SOA *Service Oriented Architecture*

URL *Uniform Resource Locator*

VLAN *Virtual Local Area Network*

WoT *Web of Things*

Sumário

1 Introdução

1.1 Motivação

Em 2014 foi implementado no câmpus IFSC-SJ, através de um projeto de pesquisa institucional (Edital nº 16/PROPPI/2013), um sistema de monitoramento de energia que permite o armazenamento de um histórico do consumo de energia para futuras análises. O sistema foi baseado em um projeto pré existente que foi adaptado para as necessidades do campus (JúNIOR et al., 2014).

Um projeto desta natureza permite identificar anormalidades de consumo e o planejamento de ações para reduzir o consumo energético.

O sistema utiliza um módulo Arduino conectado a rede cabeada através de *Power over Ethernet* (PoE). O módulo envia mensagens a um sistema supervisório implementado através de uma aplicação *Web* desenvolvida em *PHP: Hyper Text Preprocessor* (PHP) e uma base de dados MySQL. Ele provê um suporte básico para o gerenciamento das medições e seu monitoramento.

Alguns pontos foram identificados como possíveis adendos para se obter um sistema plenamente operacional. Um deles é a implantação do sistema de monitoramento na *Cloud* do campus, proporcionando facilidades de manutenção e robustez. Além disto, melhorias no sentido de geração de relatórios podem ser realizadas. Finalmente, a possibilidade de não somente realizar monitoramento mas também acionamento de cargas é uma característica desejável para um sistema desta natureza. Neste sentido, o presente trabalho foi inicialmente conduzido. Porém, foram identificadas outras possibilidades: a disponibilização da informação de consumo de energia para outros fins; o monitoramento de outras grandezas e a facilidade de integração com outras aplicações e serviços, tais como e-mail, acesso a redes sociais. Isso levou a pesquisa de modelos de Internet das Coisas, chegando ao modelo de fluxo de mensagens e do Node-RED. (LAMPKIN et al., 2012).

Neste sentido, o trabalho foi conduzido em duas frentes: (i) a implementação de melhorias no sistema e (ii) a integração do sistema supervisor com a plataforma Node-RED e o protocolo MQTT, usado na comunicação entre os nós do sistema.

1.2 Objetivos

Esse trabalho propõe incorporar melhorias e integrar o sistema supervisório de energia do campus São José a uma plataforma de Internet das Coisas baseada no modelo de fluxo de mensagens. A ideia central é proporcionar um sistema capaz de incorporar novos sensores (de grandezas diversas), facilitar a atuação em dispositivos através do protocolo MQTT e permitir a fácil integração com outros serviços.

Os objetivos específicos do trabalho são:

- Atualizar e reativar a implementação atual, melhorando a geração de relatórios no servidor;
- Realocar o servidor de registro para o IFSC-CLOUD;
- Propor um modelo de integração do sistema supervisório com o modelo de fluxo de mensagens (Node-RED);
- Incorporar o protocolo MQTT nos sensores/atuadores baseados em Arduíno.

1.3 Organização do Trabalho

Primeiramente, no Capítulo 2 é apresentado o sistema OpenEnergyMonitor, no qual o trabalho anterior foi baseado, e as tecnologias e ferramentas utilizados por ele, além de alguns detalhes do sistema OpenStack utilizados na *cloud* do campus. No Capítulo 3 é discutido o protocolo MQTT e o servidor Node-RED, explicando e justificando a escolha destas tenologias neste trabalho. No Capítulo 4 um modelo de integração do sistema existente com o servidor Node-RED e o protocolo MQTT é proposto junto a algumas aplicações exemplos. No Capítulo 5 uma pequena descrição dos procedimentos realizados na máquina virtual para a instalação e configuração dos servidores é feita, seguida pela apresentação do teste realizado com o protocolo MQTT e o o Arduino. Por fim, no Capítulo 6 é discutido os resultados deste trabalho e as deficiências observadas, além de apresentar algumas sugestões para trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo são apresentados os conceitos necessário para a compreensão do sistema supervisório de energia anteriormente implantado no IFSC-SJ. Inicialmente é apresentada uma breve descrição do OpenEnergyMonitor, no qual o sistema implantado foi baseado. Em seguida, é detalhado o aplicativo EmonCMS, com uma introdução ao modelo *Model View Controller* (MVC), o qual compõe sua estrutura, além da notação *JavaScript Object Notation* (JSON) e os bancos de dados MySQL e Redis. Na sequência é explorada a estrutura de nós de monitoramento. Finalmente é apresentado um breve resumo do Openstack e da *Cloud* do IFSC, que foram utilizados para implantação do servidor EmonCMS.

2.1 OpenEnergyMonitor

O OpenEnergyMonitor é um sistema de monitoramento com nós sensores que transmitem dados através de comunicação sem fio a uma estação base, como ilustrado na Figura 1. Ele permite o monitoramento de vários pontos de energia de forma centralizada por uma página *Web*. Sua estrutura é dividida em módulos, explicados a seguir.

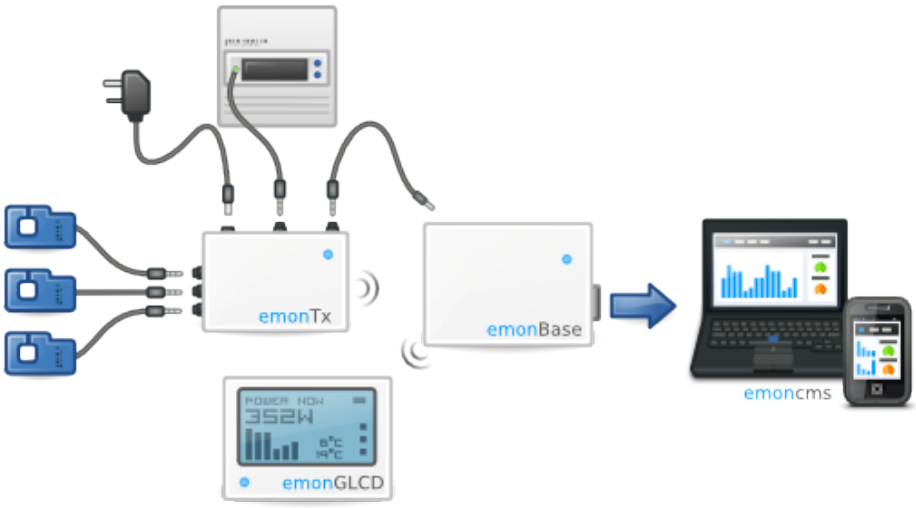

Figura 1: Estrutura do OpenEnergyMonitor (HUDSON; LEA, 2015)

O módulo EmonBase é a estação central do sistema, onde os dados obtidos pelos sensores são enviados. Ele pode ser implementado através do NanodeRF (Arduino + *clone* Ethernet), que funciona apenas como *bridge* entre os nós *wireless* e a nuvem, os enviando diretamente ao servidor remoto (EmonCMS). Também pode ser usado um RaspberryPi

com um adaptador sem fio (RFM69Pi), o qual pode ser instalado com o EmonCMS para ter-se um registro e acesso local.

O EmonTX, implementado com a plataforma Arduíno, permite formar os nós remotos do sistema. A versão V3, ilustrada na Figura 2, utiliza um microprocessador ATmega328 com o *bootloader* do Arduino Uno e pode ser customizada utilizando-se do Arduíno *Integrated Drive Electronics* (IDE) e um cabo USB-UART. Com ele pode-se mensurar: temperatura, com um sensor DS18B20; a corrente de quatro circuitos monofásicos (ou um trifásico) , utilizando transformadores de corrente; a voltagem de um circuito *Alternating Current* (AC) com adaptador AC-AC para isolamento; e um pulso óptico. Múltiplos nós podem ser conectados a estação base, mas cada um precisa de um ID *wireless* diferente.

Figura 2: EmonTX V3 (HUDSON; LEA, 2015)

O EmonCMS é um aplicativo *Web* de registro, processamento e visualização de energia, temperatura e outros dados do ambiente. Pode estar em um servidor remoto ou instalado no próprio EmonBase. Ele é estruturado conforme o modelo *Model View Controller* (MVC), descrito na Seção 2.2.3.1.

Também há outros módulos que desempenham funções mais específicas: O EmonGLCD é um *display* de *Liquid Crystal Display* (LCD) *wireless* que pode ser utilizado para visualizar os dados captados; O EmonTH é semelhante ao EmonTx, mas prioriza a medição de temperatura, umidade e do tempo de vida útil da bateria.

2.2 O servidor EmonCMS

2.2.1 Estrutura

Como a maioria das aplicações *Web* atuais, a arquitetura do EmonCMS utiliza-se de código *Hyper Text Markup Language* (HTML), *Cascading Style Sheets* (CSS) e Javascript para compor sua interface com o usuário. O cliente se comunica com a *Application Programming Interface* (API) do servidor através de solicitações em *Asynchronous Javascript and XML* (AJAX), trocando mensagens em formato JSON. Então a API HTTP do servidor aciona os modelos internos para efetuar funções como registro de dados, processamento e validação. O equipamento de monitoramento de energia se conecta diretamente ao API do servidor.

A estrutura de diretórios do EmonCMS é ilustrada na Figura 3 a seguir:

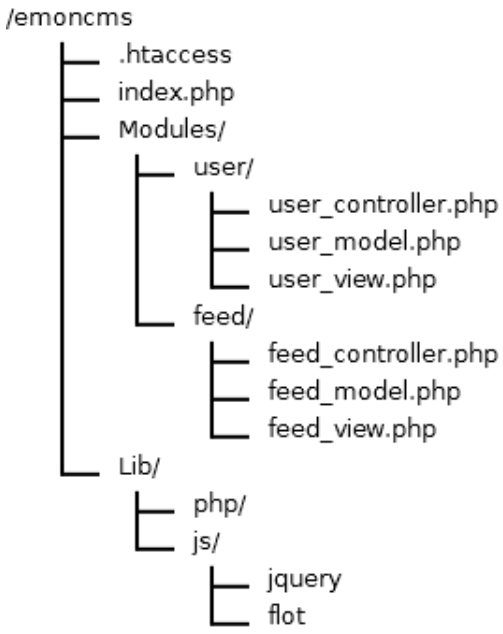

Figura 3: Estrutura de diretórios do EmonCMS.(HUDSON; LEA, 2015)

Todo o tráfego deve primeiro passar pelo index.php. Para isso configura-se o arquivo .htaccess (Apache) de forma que qualquer requisição HTTP que não apontar para um arquivo, caminho ou *link* simbólico seja enviada ao index.php. Esse então pode verificar através da *Uniform Resource Locator* (URL) (em formato JSON) qual módulo, ação e formato foi requisitado.

O EmonCMS também é dividido em módulos, a fim de permitir que funções possam ser desenvolvidas individualmente. Cada módulo pode conter as três partes da arquitetura MVC, modelo, visão e controlador, além de um *schema* de banco de dados e um arquivo de configurações do menu. Os módulos *Feeds* e *Inputs* são os principais módulos do EmonCMS e serão descritos a seguir.

2.2.2 Processamento de entradas

Antes de visualizar os dados enviados pelos sensores, é necessário convertê-los para um formato mais agradável. Porém, se os dados fossem armazenados sem nenhum tratamento, seria necessário processá-los sempre que fossem visualizados, e isto demoraria muito. Para evitar longas esperas antes de cada visualização, os dados são tratados antes de serem armazenados, o que requer a separação dos dados brutos de entrada (*inputs*) dos registros no banco de dados (*feeds*). Esta separação entre *inputs* e *feeds* permite que se manipule os dados recebidos, resultando no arquivamento e visualização de dados muito mais detalhados. (LEA, 2015)

É vinculado a cada *input* um ou mais processos, que podem armazenar o resultado em mais de um *feed*. A configuração dos processos é feita pela interface do próprio aplicativo. Alguns dos processos disponíveis, dado a versão 8.4 do EmonCMS, são:

- *Log to feed* - Registra o valor do *input* no *feed*.
- *Wh Accumulator* - Contador de watts/hora total (usado com um EmonTX configurado para enviar kW).
- *x e +* - Realiza operação do *input* por um valor fixo e envia o resultado para o próximo processo (escala e *offset*).
- *Power to kWh* - Transforma potência em kW por hora.
- *Power to kWh/d* - Transforma potência em kWh por dia.
- *Wh increments to kWh/d* - Transforma incrementos de watt/hora em um registro de kW/d.
- *kWh to Power* - Transforma kW por hora em potência média.
- *kWh to kWh/d* - Transforma kW por hora kWh por dia.
- *x,-,+ e / input* - Realiza operação do *input* pelo valor de outro *input* e envia o resultado para o próximo processo.
- *input on-time* - Conta o tempo por dia em que uma entrada está ativa (para energia solar).
- *Total pulse count to pulse increment* - Contador de pulsos.
- *x,-,+ e / feed* - Realiza operação do *input* pelo valor atual de um *feed* e envia o resultado para o próximo processo.

Os processos são executados em ordem, podendo se manipular um *input* e enviar o resultado para o processo seguinte.

2.2.3 Tecnologias usadas no EmonCMS

2.2.3.1 O modelo MVC

O MVC, usado primeiramente pelo Smalltalk e popularizado pelo Java, é a arquitetura mais utilizada em aplicações *Web* hoje. Ele separa o código de uma aplicação em três partes, descritas brevemente a seguir:

- Modelo: É a parte responsável por todas as funções lógicas da aplicação, além do armazenamento de dados e entidades.
- Visão: Encarregado de formatar e apresentar os dados recebidos do modelo.
- Controlador: Faz o intermédio entre o modelo e a visão, recebendo os pedidos do cliente, requisitando dados do modelo, os enviando a visão e encaminhando a página HTML resultante de volta ao cliente.

A Figura 4 mostra as interações e dependências entre as partes do MVC:

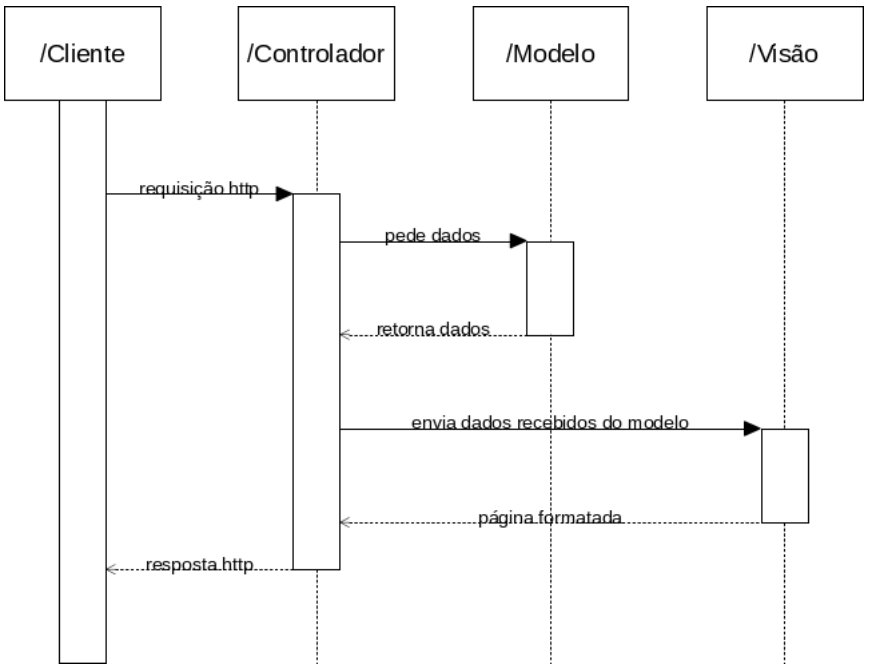

Figura 4: Interação entre os módulos do MVC (PHP-HTML, 2015).

Usualmente um aplicativo *Web* é acessado por uma página principal (index.php), que irá invocar o controlador e enviará informações sobre a requisição HTTP. O controlador então a analisa, invocando por sua vez o modelo para realizar as operações e obter os dados requisitados. Após receber a resposta do modelo, os dados são encaminhados à visão, a fim de construir a página HTTP que será visualizada pelo usuário. Por fim, o controlador envia a página construída pela visão de volta ao usuário.

2.2.3.2 JSON

O JSON, um subconjunto da linguagem de programação JavaScript, é um formato para troca de dados, projetado para ser fácil de analisar e gerar seu código. Ele independe da linguagem de programação das aplicações que utilizam-na, mas usa convenções que a torna familiar para programadores de linguagens da família C, como C, C++, C#, Java, JavaScript, Perl, Python e outras.

O JSON possui duas estruturas, utilizadas em praticamente todas as linguagens de programação: conjuntos de pares de nome e valor (*object, record, struct*) e listas ordenadas de valores (*array, vector, list*). Elas utilizam, respectivamente, as seguintes sintaxes: {nome1:valor1,nome2:valor2} e [valor1,valor2,valor3]. Os valores podem ser uma *string*, um número, *true, false, null* ou um objeto e uma *array*(JSON, 2015).

2.2.3.3 O servidor MySQL

O MySQL é um sistema de gerenciamento de banco de dados de código aberto que utiliza a linguagem *Structured Query Language* (SQL) como sua interface. Ele é utilizado por milhares de usuários em todo mundo por ser rápido, confiável, escalável e fácil de usar(MYSQL, 2015).

2.2.3.4 Redis

Redis é um banco de dados chave-valor de código livre. Ele funciona como um *cache*, fazendo os registros imediatos em memória, realizando despejos em disco periodicamente. Essa função de persistência pode ser desabilitada, se for preciso apenas de um banco de dados de acesso rápido. Assim ele pode ser usado em conjunto com outro banco de dados, como é feito com o MySQL no EmonCMS(REDIS, 2015).

2.3 EmonTX

2.3.1 Monitoramento com transformador de corrente

O *firmware* padrão do kit EmonTx V3 (emonTxV3_RFM12B_Discrete Sampling) é disponível no Github[1] do OpenEnergyMonitor. Ele foi construído para uso geral, sendo capaz de fazer a leitura com um adaptador AC, CTs (transformadores de corrente) e um sensor de temperatura, além da detecção e controle da bateria. Ele se conecta ao servidor EmonCMS utilizando a placa Wi-Fi RFM12B, o qual faz parte do kit padrão do EmonTx V3. Esquemas estão disponíveis no repositório, sendo a maioria destinados a sistemas monofásicos. O esquemático do EmonTx V3 pode ser visualizado no anexo C.

[1] https://github.com/openenergymonitor/emonTxFirmware/tree/master/emonTxV3

2.3.2 Tecnologia Usada no EmonTX: o Arduino

O Arduino é uma plataforma de baixo custo e de código aberto para prototipagem de sistemas eletrônicos baseados em uma placa com um microcontrolador simples. Com ele é possível criar um sistema capaz de obter informações de diversas origens e usá-los para controlar diversos equipamentos. A linguagem de programação utilizada é uma implementação do Wiring, e se assemelha muito ao C ou C++(ARDUINO, 2015).

2.4 OpenStack e a Cloud do Campus

O OpenStack é um sistema operacional em larga escala capaz de gerenciar os recursos computacionais, de armazenamento e de rede de uma infraestrutura de TI, permitindo que usuários possam controlar e alocar todos esses recursos através de uma interface *Web*, conforme ilustra a Figura 5.

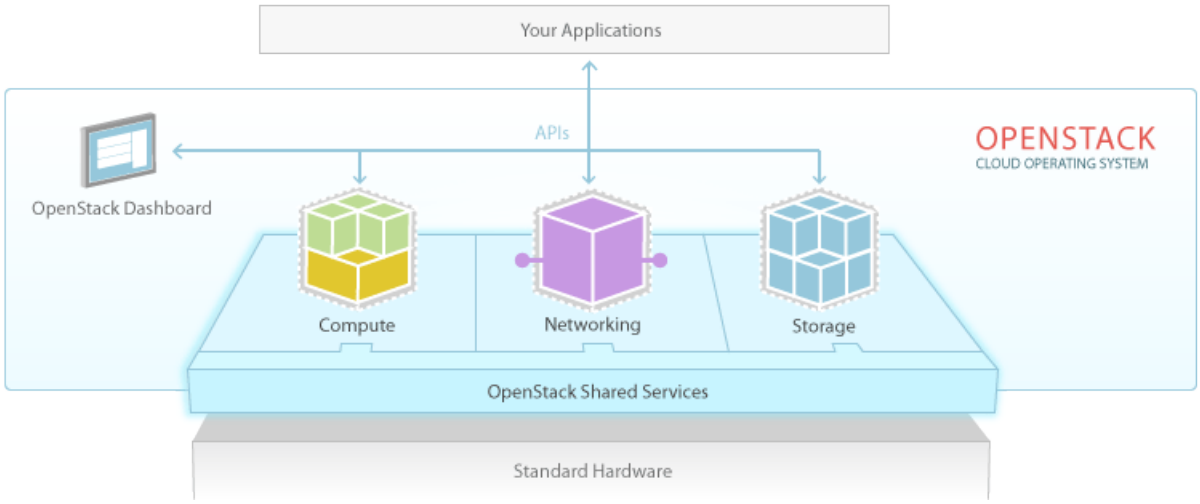

Figura 5: Diagrama da estrutura do Openstack(OPENSTACK, 2015).

Ele permite que os recursos disponíveis sejam alocados por demanda, melhorando o aproveitamento do sistema e facilitando o seu gerenciamento. Este sistema foi utilizado para a implantação do esquema de integração EmonCMS-Node-RED proposto neste trabalho.

No IFSC o sistema OpenStack (chamado de IFSC-CLOUD) está estruturado conforme a Figura 6:

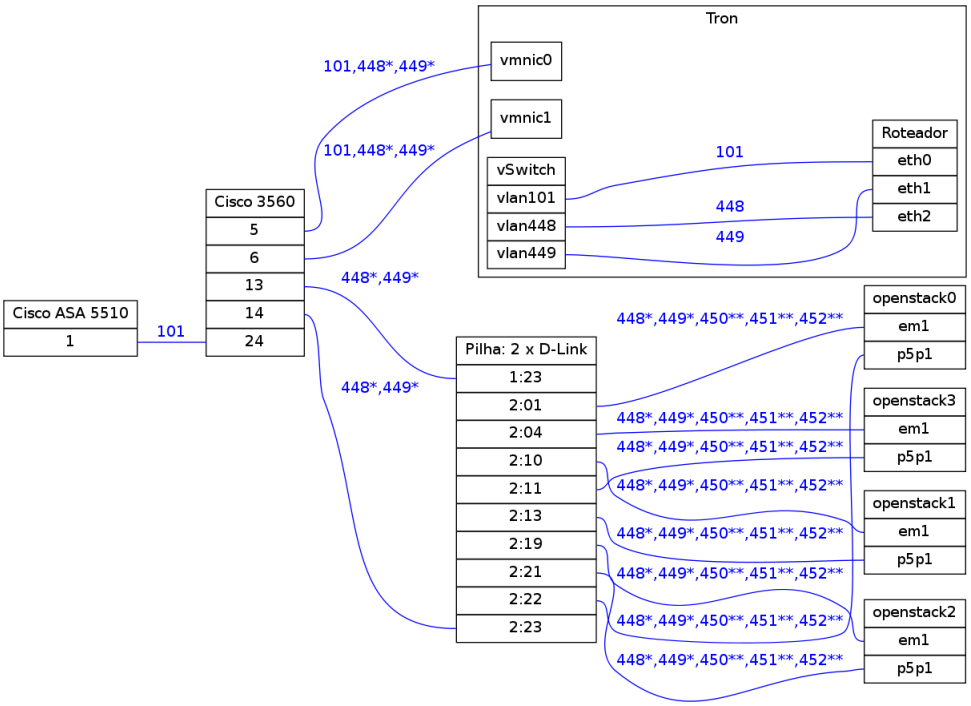

Figura 6: Estrutura atual da rede no IFSC-CLOUD(TORRESINI, 2015).

As *Virtual Local Area Network* (VLAN)s configuradas são descritas a seguir:

- 101: acesso, endereços públicos (200.135.233.252/30).
- 448: VMs, endereços públicos (200.135.233.0/25).
- 449: acesso remoto a virtualizadores, endereços públicos (200.135.233.192/28).
- 450: gerência interna da nuvem, endereços privados (RFC 1918 e RFC 4193).
- 451: sistema de arquivos distribuído, endereços privados.
- 452: interface de tunelamento entre serviços de processamento e de rede da nuvem, endereços privados. Obs.: Asteriscos (*) indicam tráfego com etiqueta 802.1q. Asteriscos duplos (**) indicam tráfego com etiqueta 802.1 e interno do ambiente (*switch* e virtualizadores).

2.5 Conclusão

Neste capítulo foi realizada uma breve revisão da tecnologia usada no sistema supervisório de energia do IFSC-SJ, tal como implementado em 2014. Também foi brevemente revista a Cloud do IFSC-SJ, sobre a qual foi implementado o sistema proposto neste trabalho.

Na sequência são descritos conceitos associados a programação em fluxo e a plataforma Node-Red que foram usados para a integração com o sistema supervisório inicialmente implantado.

3 Internet das Coisas

A estrutura do projeto OpenEnergyMonitor permite o registro das medições de múltiplos sensores, porém não oferece a capacidade de acionamento e automação de equipamentos remotos. Neste sentido, um dos focos deste trabalho foi pesquisar e identificar um modelo e plataforma adequado a finalidade de controle/acionamento, além de permitir a rápida integração de outros sensores e atuadores. O modelo de *flow programming* e a plataforma Node-RED mostraram-se interessantes, para esta finalidade, permitindo também integrar facilmente o *emoncms*.

Outro ponto foi que mesmo sendo possível implementar uma solução que utilize mensagens HTTP, tal como utilizado no EmonCMS do sistema supervisório de energia, esse não é o método mais adequado para aplicações de controle. Além de requerer recursos de rede e de hardware consideráveis, o HTTP não provê suporte para comunicações multiponto. No caso do sistema supervisório, cada nó do sistema teria que ser configurado como um servidor HTTP, o que dificultaria a instalação de novos nós. O uso do protocolo MQTT soluciona esses problemas podendo ser usado de forma robusta para o controle de dispositivos.

Essas duas tecnologias fazem parte do mundo da Internet das Coisas e este capítulo será dedicado a descrição das mesmas.

3.1 Introdução à Internet das Coisas

A *Internet of Things* (IoT) foi definida pela ITU-T (Y.2060, 2012) como uma infraestrutura global para a sociedade da informação. Ela permite o fornecimento de serviços avançados, através da interconexão de coisas (virtuais e físicas) baseadas em tecnologias de informação e comunicação interoperantes.

As coisas pertencem ao mundo físico podendo ser sensores e atuadores interconectados em rede, tais como robôs industriais, equipamentos elétricos e coisas físicas em geral. Já as coisas virtuais existem nos sistemas de informação, podendo ser armazenadas, processadas e acessadas (Y.2060, 2012). Uma coisa do mundo físico pode ser mapeada em uma ou mais coisas do mundo da informação.

A ideia que permeia a Internet das coisas é a possibilidade de conectar e acessar tudo que pode ser de interesse do homem. Por exemplo, uma geladeira, um micro-ondas, paredes e o que se possa imaginar. As aplicações IoT podem incluir *e-Health*, casas inteligentes, *smart-grids* entre outras.

3.2 Programação orientada a Fluxos

Existem duas principais abordagens utilizadas atualmente pelas comunidades científicas e industrias para o desenvolvimento de aplicações voltadas a internet das coisas: a *Web of Things* (WoT) e a *Service Oriented Architecture* (SOA).

A WoT (GUINARD; TRIFA; WILDE, 2010) se baseia na estrutura existente da *Web*, utilizando-se das normas e conhecimentos de desenvolvimento de aplicações *web*. Porém ela não define um paradigma de desenvolvimento específico, deixando essa escolha ao desenvolvedor. Por exemplo, a abordagem REST-full pode ser utilizada. Os sensores são representados através de URIs. Clientes REST seguem *links* para recursos tal como em *browsers* da Internet.

A SOA introduz camadas de abstração de dispositivos para tratar da diversidade de dispositivos além de habilitar a integração da estrutura da IoT utilizando-se de técnicas de programação orientada a objetos, o paradigma de desenvolvimento de *software* mais popular atualmente.

Porém essas abordagens nem sempre são as mais apropriadas, como afirmam Lobunets e Krylovskiy:

> Apesar do sucesso da programação orientada a objeto no domínio mais amplo do desenvolvimento de *software*, nossa experiência sugere que às vezes ela se torna um fardo: aplicar seus princípios diretamente no desenvolvimento de aplicações orientadas a dados para ambientes inteligentes é desafiador e muitas vezes resulta em uma diferença significativa entre a arquitetura inicial e a implementação em *software*. Procurando por uma abordagem alternativa capaz de uma melhor realização das ideias de projeto, descobrimos a programação baseada em fluxo, que é um subconjunto de uma abordagem de fluxo de dados mais geral para construção de *software*(LOBUNETS; KRYLOVSKIY, 2014).

Na programação orientada à fluxos, os aplicativos são construídos como uma rede de componentes reusáveis. Cada componente possui um número de entradas e saídas que recebem ou enviam dados e podem ser construídas com uma linguagem de alto-nível ou um outro fluxo. Desta forma, uma aplicação orientada a fluxo pode ser considerada como dirigida ou guiada pelos dados (*data driven*).

O motor de execução da rede de componentes cria um processo para cada componente e estabelece conexões entre portas de entradas e de saída utilizando-se de *buffers* de tamanho controlado (LOBUNETS; KRYLOVSKIY, 2014).

3.3 Node-RED

O Node-RED proporciona um ambiente gráfico de programação de fluxo de mensagens. Trata-se de uma plataforma web construída sobre a estrutura Node.js. O Node.js se utiliza da linguagem de programação Javascript para o desenvolvimento de aplicações do lado do servidor.

O Node-RED permite construir um gráfico (fluxo) composto de nós, os quais podem ser mensagens de entrada e saída ou uma lógica. As mensagens são passadas pelos nós, podendo ser processadas ou transformadas. Alguns nós também podem ser usados para armazenar dados.

A Figura 7 mostra uma visão da estrutura do Node-RED. Pode-se observar à direita a estrutura no lado servidor constituída por um ambiente em tempo de execução do Node.js e um motor do Node-RED. É no servidor onde se executam os nós que tratam os fluxos de mensagens. No lado esquerdo da figura pode-se observar o lado cliente (*browser*), que proporciona uma editor gráfico que permite selecionar nós de uma biblioteca e interconectá-los.

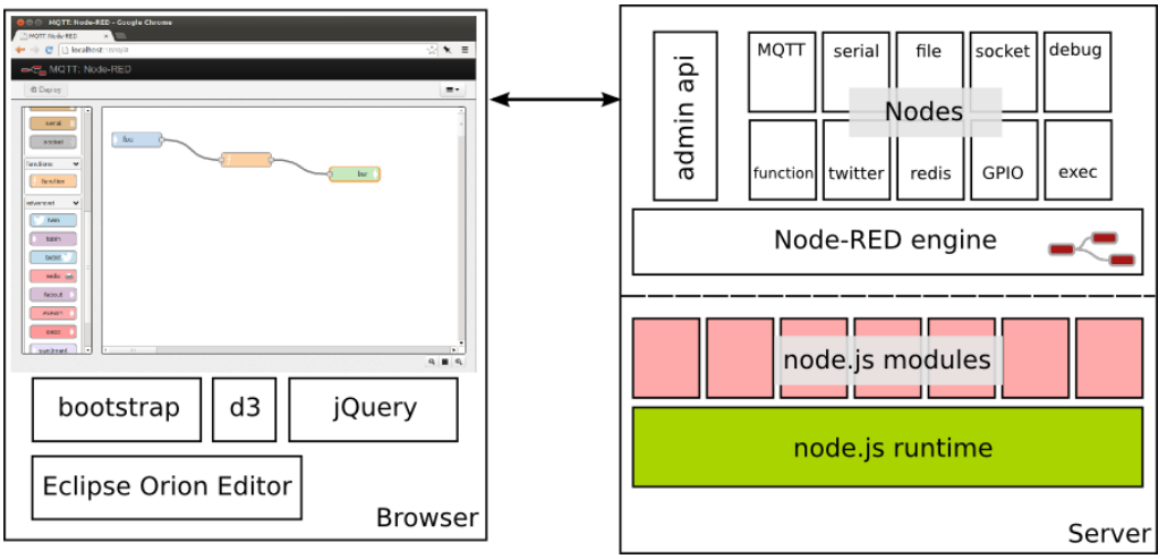

Figura 7: Estrutura do Node-RED. (WENDE, 2016)

Cada fluxo é armazenado usando JSON, sendo possível copiar fluxos inteiros ou partes copiando o texto JSON de outro projeto. Por rodar sobre Node.js, ter acesso as suas bibliotecas, e permitir a criação de novos nós pela comunidade (compostos de arquivos HTML e Javascript), o Node-RED permite grande flexibilidade para integrar diferentes tecnologias. Essa flexibilidade e facilidade de uso o tornou uma opção interessante para implementar a parte ativa do sistema supervisório de energia que é objetivo deste trabalho.

O projeto Node-RED é hospedado no repositório Github (O'LEARY; CONWAY-JONES, 2016) e está disponível pelo gerenciador de pacotes do Javascript, o Npm. Uma aplicação Node-RED simples é mostrada na Figura 8. Pode-se observar a programação visual em que um nó é responsável pela recepção e processamento de uma mensagem do *tweeter* (#LED) e que dispara um temporizador que por sua vez dispara uma mensagem para acionar o pino 12 de uma placa Raspberry PI. Não é mostrado a programação em Javascript que pode ser inserida em cada nó.

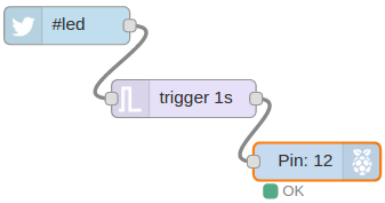

Figura 8: Aplicação Simples Node-Red. (LEA; BLACKSTOCK; CALDERON, 2016)

O Node-RED possui um módulo de protocolo MQTT capaz de interagir com o servidor Mosquitto, um mediador MQTT, permitindo desta forma a comunicação com placas e sistemas que se comunicam com este protocolo. Neste projeto, implantou-se o MQTT em uma placa Arduíno. O MQTT é descrito na sequência.

3.4 MQTT

O MQTT é um protocolo de comunicação de máquina para máquina usado em aplicações de Internet das Coisas. Ele foi criado pela IBM e com padrão mantido pela OASIS. Projetado para ser leve, ele utiliza o modelo de publicar/subscrever, no qual dispositivos podem se inscrever em um determinado tópico e receber as mensagens publicadas no mesmo. Sua estrutura define um servidor mediador (*broker*) e dois tipos de cliente: subscritor (*subscriber*) e o publicador (*publisher*)(OASIS, 2015).

3.4.1 Comparação entre HTTP e MQTT

A seguir é apresentado uma comparação rápida entre o protocolos MQTT e HTTP na tabela 3.4.1, seguida de uma descrição de cada item:

Tabela 1 – Comparação entre HTTP e MQTT (LAMPKIN et al., 2012)

	MQTT	HTTP
Orientação do Design	centrado a dados	centrado a documentos
Modelo	publicar/subscrever	cliente/servidor
Complexidade	simples	complexo
Tamanho das mensagens	pequeno, com um *header* de 2 B	grande, por possuir descritores em texto
Níveis de Serviço	3 configurações	o mesmo para todas as mensagens
Bibliotecas	C (30 kB) e Java (100 kB)	Depende da aplicação, mas usualmente grande
Distribuição de dados	1 para 0, 1 para 1 e 1 para n	1 para 1

- Orientação: HTTP: Sendo orientado a documentos, ele suporta o padrão *Multipurpose Internet Mail Extensions* (MIME), o que não é usualmente necessário em dispositivos com capacidade limitada; MQTT: É orientado a dados, transmitindo dados como conjuntos de bytes sem se importar com conteúdo.

- Modelo de mensagens: HTTP: Usa o modelo cliente/servidor, o qual é simples e poderoso, porém o cliente precisa conhecer o endereço do dispositivo ao qual ele deseja conectar-se; MQTT: Usa o modelo de publicar/subscrever, onde o cliente não precisa conhecer o destinatário , precisando apenas se preocupar com o conteúdo a ser entregue ou recebido.

- Complexidade: HTTP: É relativamente complexo, possuindo diversos códigos e métodos de resposta; MQTT: Mais simples, possuindo apenas os tipos de mensagens mais importantes para desenvolvedores.

- Tamanho das mensagens: HTTP: Utiliza um formato com texto legível, o que acarreta em (*headers*) longos; MQTT: É projetado para dispositivos de pouca capacidade, incluindo apenas as funcionalidades necessárias para suporta-los. O tamanho de seu menor pacote é de 2 bytes.

- Qos: HTTP: Não possui mecanismos de repetição ou QoS; MQTT: Suporta três níveis de QoS.

- Bibliotecas extras: HTTP: Não necessita de nenhuma biblioteca por si próprio, porém necessita de bibliotecas ao utilizar serviços (*web*) com arquiteturas do tipo SOAP ou RESTful; MQTT: Requer poucas bibliotecas, com um total de 30 kB para o cliente em C e 100 kB para o cliente em Java.

- Distribuição de dados: HTTP: É ponto-a-ponto e funcionalidades de distribuição precisam ser adicionadas externamente; MQTT: Suporta modelos de distribuição de 1 para 0, 1 para 1 e 1 para muitos.

3.4.2 Modelo Publicar/Subscrever

Com o modelo de publicar/subscrever usado pelo MQTT, é possível conectar fontes de informação (*publisher*) com consumidores (*subscriber*) por intermédio de um servidor (*broker*). Diferente da estrutura ponto-a-ponto tradicional, no modelo publicar/subscrever o dispositivo ou aplicativo que envia dados não precisa saber nada sobre quem recebe e vice-versa. O publicante simplesmente envia uma mensagem com um identificador de um tópico e então o servidor mediante distribui a mensagem para todos os dispositivos ou aplicações que se inscreverão naquele tópico.

O tópico define o conteúdo da mensagem ou o tipo de assunto que a mensagem trata. A importância dos tópicos dá-se por que, diferente de sistemas ponto-a-ponto onde as mensagens são enviadas para endereços de destino específicos, as mensagens são distribuídas de acordo com as escolhas de tópicos dos inscritos. Ao se inscrever em um determinado tópico, o inscrito está apto a receber qualquer mensagens publicada neste tópico de qualquer publicante.

Os tópicos são organizados hierarquicamente em árvores, utilizando-se da barra (/) para criar subtópicos no *string* do tópico.

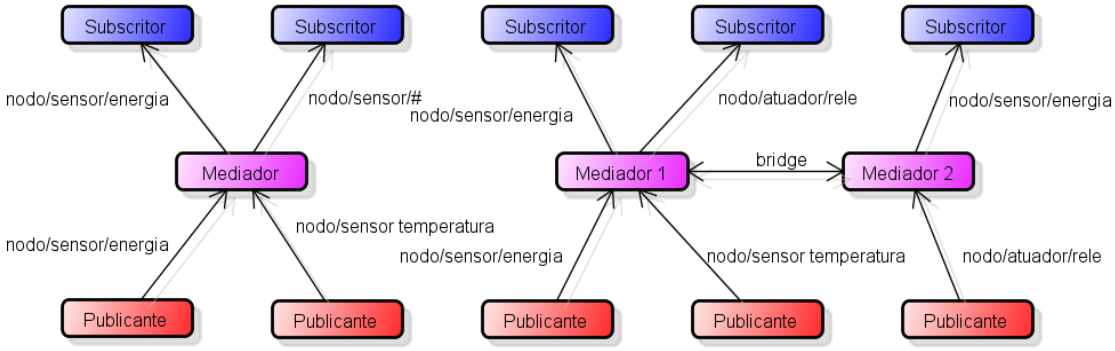

Figura 9: Dois exemplos de configurações publicar/subscrever (LAMPKIN et al., 2012)

O lado esquerdo da Figura 9 mostra um modelo publicar/subscrever simples, onde os publicantes e subscritores enviam e recebem mensagens enquanto o servidor mediante se encarrega de rotear as mensagens baseando-se nos seus respectivos tópicos. No lado direito temos uma configuração mais avançada, onde dois servidores mediantes trocam informações internamente através de uma *bridge*, o que permite que clientes conectados ao servidor 1 recebam mensagens publicadas no servidor 2. Isso simplifica a manutenção do sistema de distribuição de mensagens, o que leva a uma maior escalabilidade do que o sistema ponto-a-ponto tradicional.

3.4.3 Conceitos do MQTT

Nesta seção serão apresentados alguns conceitos do protocolo MQTT.

- Identificador do cliente: Cada cliente possui um identificador de 23 bytes único para cada cliente conectado, o qual é normalmente gerado a partir de seu endereço *Media Access Control* (MAC);
- Qualidade de Serviço: O MQTT possui *Quality of Service* (QoS) em três níveis: 0, onde o mediador/cliente envia a mensagem uma vez, sem confirmação; 1, onde o mediador/cliente envia a mensagem pelo menos uma vez, com confirmação; 2, onde o mediador/cliente envia a mensagem exatamente uma vez, utilizando um *handshake* de quatro passos.
- Retenção de publicações: Publicações para um determinado tópico podem ser retidas para serem enviadas para novos inscritos. Cada mensagem possui um atributo para retenção, que determina se a mensagem deve ser retida ou não para futura entrega.
- Filtros de tópico: Dois caracteres podem ser usados pelos subscritos para filtrar as mensagens recebidas. Com o caractere # é possível receber todas as mensagens do

mesmo nível e de níveis inferiores da hierarquia (nó/# receberia tanto nó/sensor quanto nó/sensor/energia). Com o caractere + o subscrito recebe todas as mensagens de mesmo nível hierárquico (nó/# receberia nó/sensor mas não nó/sensor/energia)

- Seções limpas: Ao se conectar, o cliente MQTT identifica a conexão com o identificador do cliente e o endereço do servidor e esse por sua vez verifica se existe alguma sessão para está conexão. Se uma sessão existir e o valor do parâmetro *cleanSession* for *true*, as informações de sessão no cliente e no servidor são limpas. O valor padrão de *cleanSession* é *true*.

- Publicações de último desejo e testamento: Se uma conexão MQTT é terminada inesperadamente, é possível configurar uma mensagem testamento a ser publicada em um tópico. O conteúdo e tópico da publicação deve ser predefinido. Essa configuração deve ser feita antes da conexão com o cliente.

3.5 Conclusão

Neste capítulo foram descritos aspectos da programação em fluxo e da plataforma Node-RED que se utiliza deste conceito para implementar aplicações baseadas em fluxo de mensagens e que podem ser usadas na Internet das Coisas. Também foi apresentado o protocolo MQTT, uma alternativa interessante que pode ser usado com tais aplicações.

No capítulo seguinte é apresentado um esquema de integração do Node-RED com o sistema EmonCMS usado no Sistema Supervisório de Energia implantado no IFSC Câmpus São José. Aplicações implementadas sobre este sistema também são descritas.

4 Modelo de Integração

Neste capitulo será apresentada a reestruturação do sistema supervisório de energia do IFSC câmpus SJ, detalhando como é feita a inclusão do protocolo MQTT e a plataforma Node-RED.

Aplicações desenvolvidas sobre este novo sistema são descritas, mostrando as potencialidades do sistema no sentido de incorporar novos sensores e atuadores conforme a filosofia de Programação em Fluxo e da Internet das Coisas.

4.1 Sistema Legado

O sistema supervisório de energia do IFSC câmpus SJ foi implementado conforme o modelo do OpenEnergyMonitor descrito na seção 2.1 do capítulo 2.

Neste sistema mostrado na Figura 10, o servidor EmonCMS fornece uma interface via HTTP-JSON através da qual um cliente (nó) pode postar dados em uma base de dados mySQL. O cliente, neste sistema, é um Arduíno que lê dados periodicamente de um medidor de energia e então realiza as postagens (mensagens 1 e 2) conforme o padrão, informando no texto JSON o identificador de nó e o valor lido. Um usuário pode a qualquer momento acessar o sistema via *browser* (HTTP-JSON)) para obter gráficos em tempo real (e históricos) sobre o consumo de energia (mensagens 3 e 4).

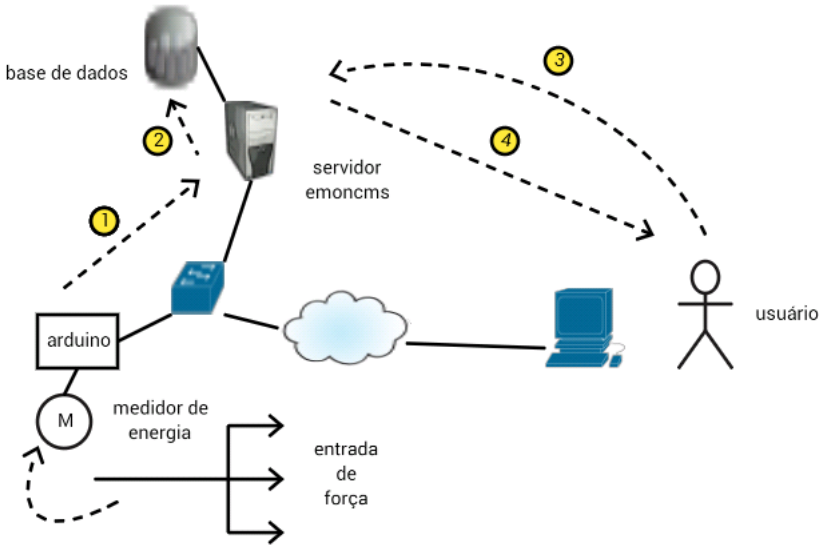

Figura 10: Sistema Legado.

4.2 Integração dos Sistemas Emoncms e do Node-RED

Um dos desafios deste trabalho foi a inclusão da plataforma Node-RED e do protocolo MQTT sem impactar a forma como foi implementado o sistema legado. Desta forma, se torna possível a continuidade de uso da geração de relatórios e acesso a gráficos em tempo real do Emoncms.

Como detalhado na seção anterior, o envio de mensagens pelo Arduíno era feito diretamente ao servidor EmonCMS, que por sua vez armazenava as medições em seu banco de dados. Através do processamento de entradas oferecida pelo servidor EmonCMS era possível apenas criar aplicações simples e passivas, já que ele não possui suporte para tratamento de eventos.

Com a inclusão do servidor Node-RED torna-se possível a criação de aplicações ativas complexas e, junto ao servidor MQTT mosquitto, utilizar um protocolo mais adequado tanto para leitura de sensores como para o acionamento de cargas. Além disto, o uso da abordagem Programação com Fluxos se torna possível. A Figura 11 mostra como a inclusão deste dois servidores é feita.

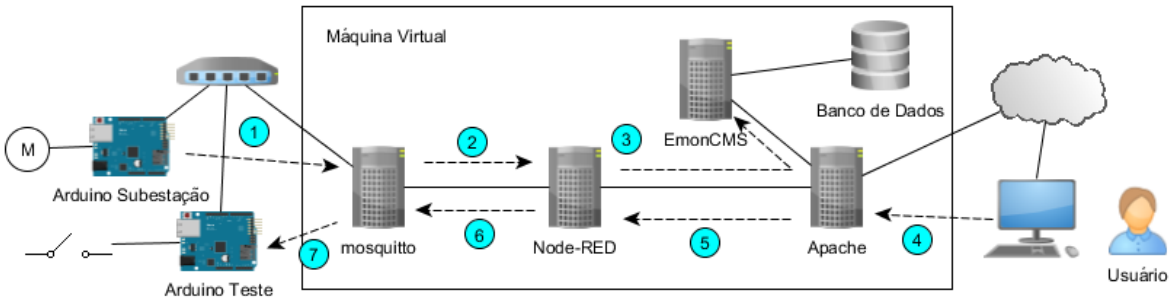

Figura 11: Sistema proposto.

Neste modelo, em vez de enviar as mensagens diretamente ao servidor EmonCMS, o Arduíno as envia primeiramente ao servidor Node-RED através de mensagens MQTT (mensagem 1 e 2). Isso permite que as medições estejam disponíveis para processamento nos fluxos do Node-RED e que as mensagem sejam traduzidas para HTTP de modo que possam ser reencaminhadas ao servidor EmonCMS (mensagem 3).

O sistema também permite que um Arduíno de teste receba comandos do usuário através de mensagens MQTT. O usuário envia um pedido HTTP com parâmetros para o servidor Node-RED (mensagem 4 e 5) que o repassa através de uma publicação em um tópico no servidor mosquitto (mensagem 6). Então o Arduino teste, que se subscreveu a esse tópico previamente, recebe a mensagem publicada (mensagem 7). O reencaminhamento das mensagens do Arduíno e o tratamento dos comandos do usuário são apresentados em mais detalhes nas seções 5.2 e 5.3, respectivamente.

Com esse modelo o servidor EmonCMS torna-se exclusivamente o gerenciador do banco de dados. O Node-RED cumpre o papel de *gateway* para o EmonCMS e faz o tratamento de resposta a eventos. O servidor MQTT mosquitto fica responsável pela comunicação com o Arduino na rede local e o Apache se encarrega da interface com o usuário.

O fluxo do Node-RED que faz o encaminhamento das mensagens do Arduíno para o EmonCMS é ilustrado na Figura 12:

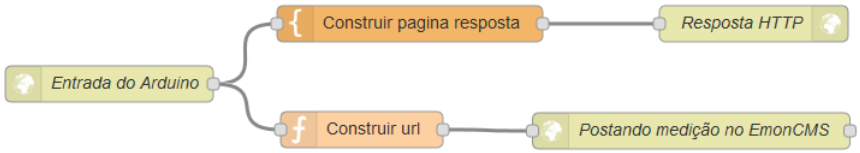

Figura 12: Fluxo Node-RED de reencaminhamento de mensagens para o EmonCMS.

O ramo superior do fluxo cria uma resposta HTTP simples para o Arduíno, enquanto no ramo inferior o URL é construído e o pedido HTTP GET contendo a medição é feito ao EmonCMS.

Uma interface também pode ser criada em conjunto com o servidor Node-RED para o controle de suas aplicações, como será mostrado em um dos exemplos a seguir.

4.3 Aplicação desenvolvida sobre o sistema

A fim de ilustrar o potencial do modelo sugerido foram criadas funcionalidades em uma aplicação desenvolvida sobre o esquema de integração proposto. A ideia é usufruir do modelo de programação em fluxo para que se possa examinar os valores de energia recebidos do nó de leitura de energia e então alertar o administrador de valores acima do norma e, se necessário, atuar no sistema desligando cargas (via protocolo MQTT). Estas funcionalidades são basicamente:

- Alerta automático de passagem de limiar de potência e atuação automática;
- Envio automático de relatórios;
- Interface de configuração de parâmetros.

Elas são descritas a seguir.

4.3.1 Atuação automática e alertas

É possível analisar as medições recebidas de um Arduíno e enviar mensagens de alerta caso esteja próximo de um limiar e efetuar o desligamento automático dos equipamentos no caso de sobrecarga. A Figura 13 mostra o diagrama de fluxo da aplicação exemplo construída:

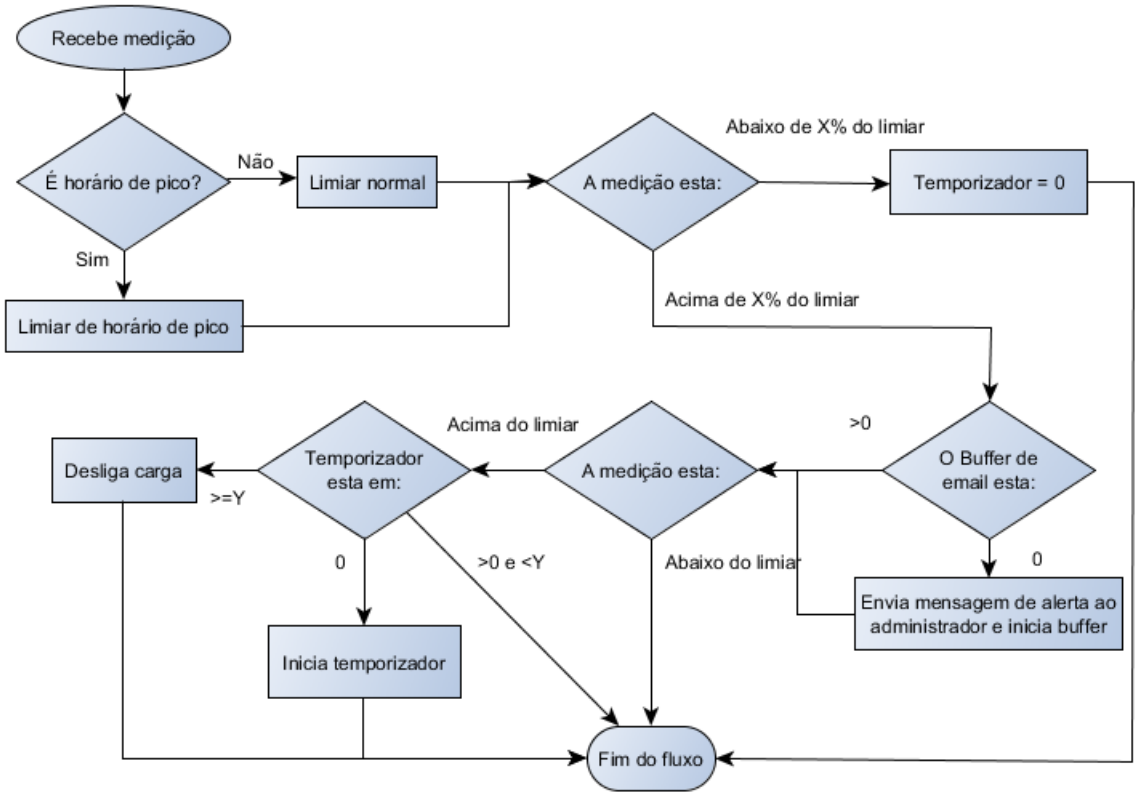

Figura 13: Diagrama de fluxo da aplicação de automatização de energia e alertas.

A cada mensagem recebida do Arduíno um fluxo desta aplicação é iniciada, sendo primeiro determinado o horário atual, a fim de determinar qual limiar de potência será utilizado. Em seguida a medição é comparada com o limiar e, caso esteja a uma certa porcentagem deste limiar, uma mensagem de alerta é enviada ao supervisor do sistema.

A fim de evitar que picos de potência desliguem a carga, a aplicação envia a mensagem MQTT de desligamento apenas depois de certo tempo em que a potência esteja superior ao limiar, iniciando um temporizador caso contrário. Este temporizador é parado se o valor de potência voltar a ser menor que a porcentagem do limiar verificada anteriormente.

A Figura 14 ilustra esse processo em um gráfico de potência por tempo das medições. O período 1 representa um pico na potência, o qual resultará em um alerta mas não no desligamento da carga. O período 2 representa um consumo que ultrapassou o limite pelo tempo determinado, o que resultará no envio da mensagem de acionamento ao Arduíno.

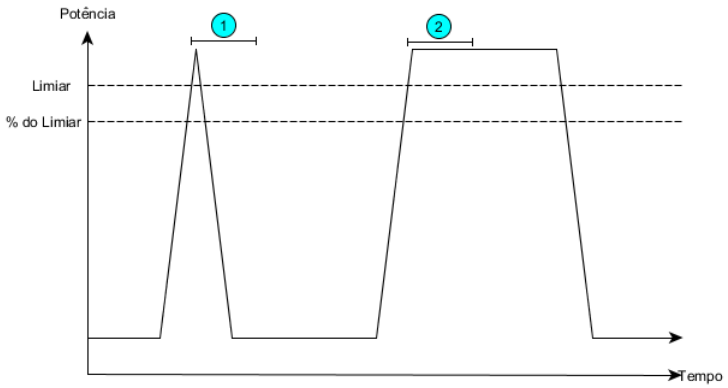

Figura 14: Gráfico de Potência x Tempo da aplicação de atuação automática.

A Figura 15 ilustra o fluxo desta aplicação na interface do Node-RED.

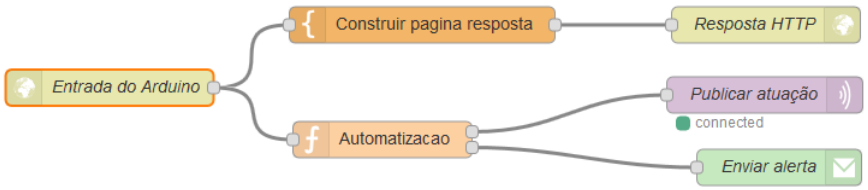

Figura 15: Fluxo Node-RED de automatização de cargas e alertas.

As porcentagem do limiar, o horário de pico, os limiares de horário e pico e horário normal e tempos de *buffer* de e-mail e de limiar ultrapassado estão disponíveis para configuração no fluxo de configuração.

4.3.2 Envio automático de relatórios

Para complementar a interface de geração de relatórios, foi construído o fluxo ilustrado na Figura 16, que pode automaticamente enviar relatórios ao administrador do sistema. Ele é iniciado por um nó *inject* de entrada programado para ser acionado ao fim de cada sexta-feira e injetar o *timestamp* atual. Utilizando esse *timestamp* um pedido é enviado ao EmonCMS através da função emoncms/feed/data de seu API para extrair os dados da ultima semana. Esse dados são então convertidos ao formato CSV e enviados por e-mail.

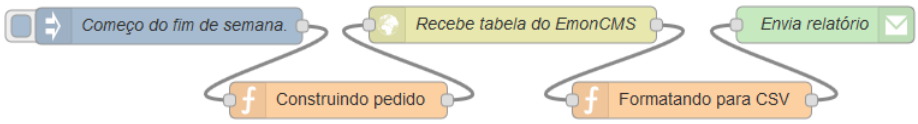

Figura 16: Fluxo Node-RED de automatização de cargas e alertas.

4.3.3 Interface de Configuração de Parâmetros

Os dois fluxos apresentados a seguir fornecem uma interface ao usuário. Ambos estão configurados para receber mensagens HTTP com parâmetros (*query*) que são utilizados em seu processamento. Suas entradas poderiam facilmente serem modificadas para criarem uma interface unificada.

O primeiro fluxo disponibiliza as variáveis utilizadas nas outras aplicações. Os parâmetros passados pelo usuário são guardados em variáveis globais disponíveis a outros fluxos. A Figura 17 ilustra o fluxo de configuração das variáveis.

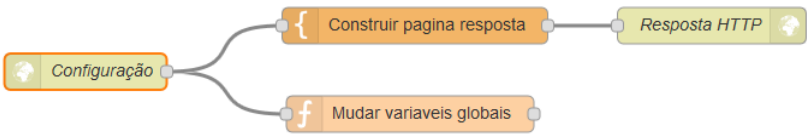

Figura 17: Fluxo Node-RED para configuração de variáveis globais.

O segundo fluxo, ilustrado na Figura 18 permite o acionamento remoto manual dos nós atuadores. Ele utiliza um pequeno bloco de função para construir a mensagem MQTT e um bloco de saída que se conecta com o servidor mosquitto e publica a mensagem.

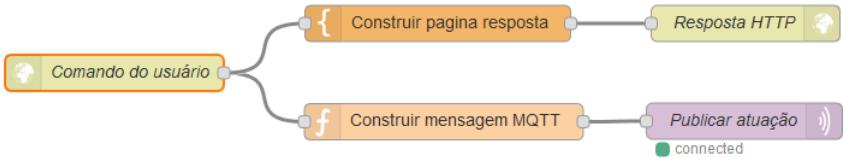

Figura 18: Fluxo Node-RED para acionamento manual de cargas.

4.4 Conclusões

Este capítulo descreveu um esquema de integração do sistema EmonCMS usado no sistema supervisório de Energia do ISFC Câmpus São José com a plataforma Node-RED. Esta plataforma permite implementar aplicações de forma flexível no domínio da Internet das Coisas. Foi incorporado neste esquema um servidor de protocolo MQTT que permite o uso de dispositivos com este protocolo.

Uma aplicação implementada sobre este sistema foi apresentada. No capítulo seguinte discute-se aspectos associados a implantação e teste deste sistema.

5 Implementação e testes

Neste capitulo é feito um rápido detalhamento dos programas instalados e configurações realizadas para o funcionamento do sistema. Também é apresentado o reencaminhamento das medições pelo servidor Node-RED e o teste realizado com o protocolo MQTT para a funcionalidade de acionamento remoto de cargas.

5.1 Implementação do servidor

Depois do servidor EmonCMS ser instalado na máquina virtual foi feito o resgate do banco de dados com as medições anteriores que estavam no servidor emoncms.org utilizando um modulo utilitário disponível no repositório Github do EmonCMS.

O servidor Node-RED foi instalado junto ao ambiente Node.js, através do gerenciador de pacotes npm. Utilizando o npm também é possível adicionar nós para serem utilizados na interface editor de fluxos. A configuração padrão do Node-RED disponibiliza a interface na porta 1880. Porém, para evitar conflitos e falhas de segurança, foi configurado um *proxy* reverso no servidor Apache para que fosse possível utilizar a porta 80 para a interface.

O servidor mosquitto, que age como mediador MQTT, foi instalado com as configurações básicas. Ele espera mensagens na porta 1883, a qual foi liberada para roteamento externo. O pacote mosquittoclient também foi instalado para permitir o envio de publicações e subscrições MQTT e facilitar testes.

5.2 Reencaminhamento das mensagens do Arduino

As mensagens HTTP enviadas ao endereço 200.135.233.25/nodered/input/post pelo Arduino na subestação iniciam a sequência de mensagens da Figura 19. O nó "Construir url"ilustrado na Figura 12 as recebe e troca /nodered/input/post por /emoncms/input/post, efetuando o reencaminhamento ao EmonCMS com um nó de pedido HTTP.

O propósito deste processo é tornar possível a análise e processamento dos valores das medições, permitindo aplicações como o acionamento automático descrito na Seção 4.3.1, além de permitir a flexibilização das mensagens enviadas: pode-se facilmente modificar o fluxo Node-RED para reencaminhar mensagens que utilizam protocolos além do HTTP.

Assim, esse reencaminhamento de mensagens permite que o gerenciamento do banco de dados das medições continue sendo feito através do EmonCMS, assim como era

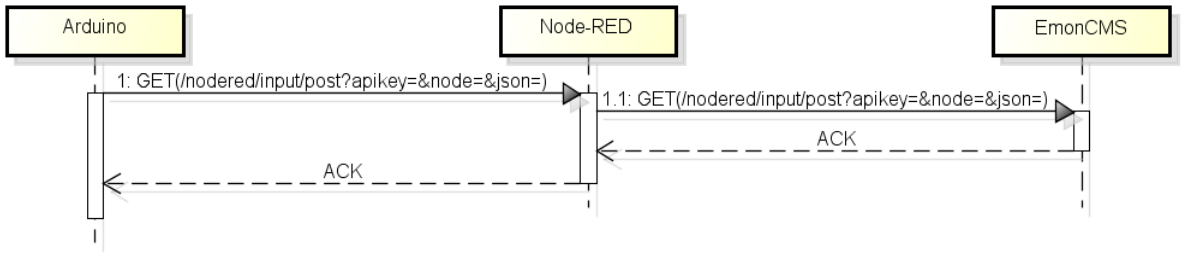

Figura 19: Diagrama de sequência das mensagens das medições do Arduino.

feito no sistema legado. Com sua interface, gráficos como o ilustrado na Figura 20 podem ser visualizados em tempo real ou através dos dados coletados.

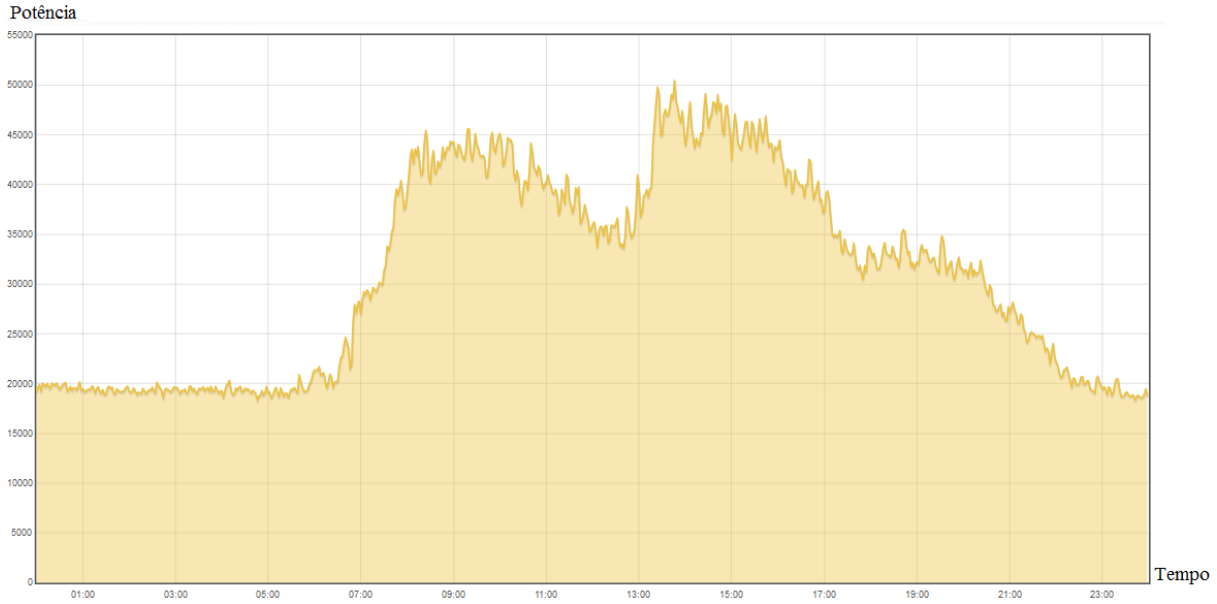

Figura 20: Gráfico de potência no tempo visualizado através da interface do EmonCMS.

5.3 Teste de acionamento com MQTT

Esse teste consistiu em fazer com que uma placa Arduíno Ethernet, ao comando de um usuário, recebe-se uma mensagem MQTT e modificasse o estado de uma de suas saídas, que nesse caso é o pino digital 9 que está conectado a um LED.

5.3.1 Troca de mensagens

A sequência das mensagens trocadas entre os servidores e a interface do usuário é ilustrada na Figura 21.

As mensagens 1, 2 e 3 são trocados quando o servidor Node-RED e o Arduíno são iniciados, para estabelecer uma conexão com o mediador e, no caso da mensagem 3, subscrever-se ao tópico relevante.

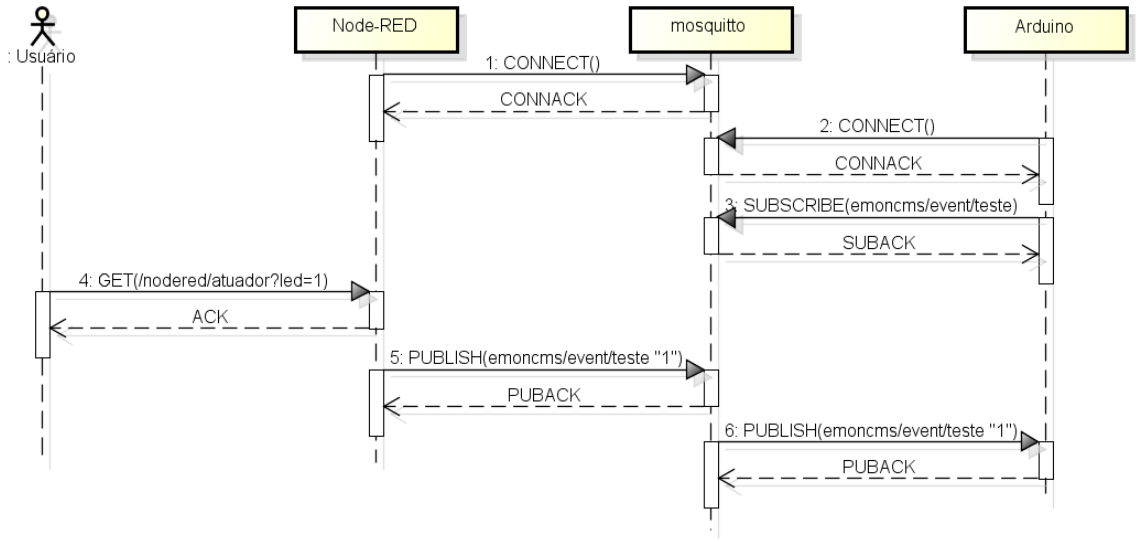

Figura 21: Diagrama de sequência das mensagens da aplicação de acionamento remoto.

Utilizando-se do fluxo mostrado na Seção 4.3.3 do Capítulo 4, o servidor Node-RED recebe a mensagem GET HTTP enviada através de um *browser* para a URL http://200.135.233.25/nodered/atuador?led=1 (mensagem 4). O fluxo responde ao Arduíno com uma página HTTP simples utilizando os nós "Construir pagina resposta"e "Resposta HTTP". O *query* HTTP led da URL é então analisado pelo nó "Construir mensagem MQTT", mostrado na Listagem 5.1 que define o tópico (msg.topic) e o corpo (msg.payload) da mensagem MQTT publicada no mediador (mensagem 5). O servidor mosquitto então repassa a mensagem ao Arduíno, que se subscreveu àquele tópico previamente (mensagem 6).

A Figura 22 mostra esse fluxo na interface Node-RED e a saída dos dois primeiros nós depois de receber o comando do usuário.

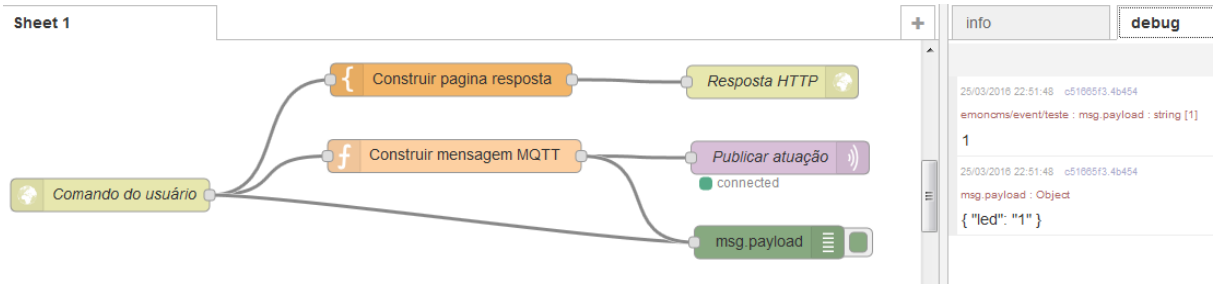

Figura 22: Fluxo de acionamento remoto com mensagens de *debug*.

Listagem 5.1 – Código do nó "Construir mensagem MQTT".

```
1 msg.topic = "emoncms/event/teste";
2 if (msg.req.query.led == "1" || msg.payload =="0") {
3     msg.payload = "1";
4 } else if (msg.req.query.led == "0") {
5     msg.payload = "0";
6 } else {
7 return null;
8 }
9 return msg;
```

5.3.2 Código do Arduino

Para implementar o protocolo no Arduíno foi utilizado a biblioteca PunSubClient.(O'LEARY, 2016) como pode-se observar na Listagem 5.2 do código utilizado no Arduíno.

Ao ser iniciado o Arduíno executa a função setup (linha 23) que configura o pino digital 9 (led) para saída. A função PubSubClient (client) é informada a porta e o endereço do servidor mosquitto e o nome da função de retorno (callback). Por fim ele inicia sua interface ethernet, obtendo um endereço IP por *Dynamic Host Configuration Protocol* (DHCP).

Após essa configuração inicial a função principal loop (linha 30), que checa se há conexão com servidor e chama a função de reconnect caso não houver. Nesta função é que o tópico é definido e a subscrição efetuada (linha 20).

Enquanto a conexão com o servidor estiver estabelecida, a função callback (linha 9) é executada sempre que a mensagem atualização do tópico for recebida e seu conteúdo analisado. Se a mensagem for "1"o pino digital 9 é levado ao estado alto e o LED acesso.

Listagem 5.2 – Código de subscriçao MQTT do Arduino.

```
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>

byte mac[] = {  0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };
IPAddress server(200, 135, 233, 25);
int led = 9;

void callback(char* topic, byte* payload, unsigned int length) {
  if ((char)payload[0] == '1') {
      digitalWrite(led, HIGH);  } else {
      digitalWrite(led, LOW);   }   }

EthernetClient ethClient;
PubSubClient client(ethClient);

void reconnect() {
  while (!client.connected()) {
    if (client.connect("arduinoClient")) {
      client.subscribe("emoncms/event/teste");  } else {
      delay(5000);  }   }   }

void setup() {
  pinMode(led, OUTPUT);
  client.setServer(server, 1883);
  client.setCallback(callback);
  Ethernet.begin(mac);
  delay(1500);  }

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

5.4 Conclusão

Neste capítulo foi feito um relato das operações realizadas na máquina virtual para a implementação dos sistema. Também foi detalhado o funcionamento do reencaminhamento de mensagens do Arduíno da subestação e o envio de comandos ao Arduíno de teste.

Na capítulo seguinte será discutido as conclusões finais do trabalho e serão feitas algumas sugestões para trabalhos futuros.

6 Conclusões

6.1 Considerações Finais

O objetivo inicial deste trabalho foi o de implantar melhorias no sistema supervisório de energia do IFSC-SJ. Entretanto, esta meta inicial foi de certa forma tornada secundária pelo fato de se ter observado a possibilidade de ampliar as potencialidades do sistema para o mundo da Internet das Coisas usando a plataforma Node-RED e o protocolo MQTT.

A inclusão destas duas tecnologias era proposta pela comunidade do projeto OpenEnergyMonitor a fim de solucionar as deficiências do sistema para aplicações de acionamento ativo. Porém, como ainda não havia um suporte oficial a essa integração, este trabalho se dedicou a criar um esquema de implementação próprio do servidor Node-RED e o protocolo MQTT para que fosse possível determinar sua capacidade.

A inclusão do servidor Node-RED e seu paradigma de programação orientada a fluxo torna o sistema muito mais flexível. Novas funcionalidades podem ser facilmente criadas e integradas ao sistema existente. A implantação do sistema na Cloud do câmpus torna o sistema mais robusto.

Já que encontrar um método para acionamento remoto era apenas um dos objetivos deste trabalho, não foi possível contemplar as questões de segurança do protocolo MQTT e do servidor Node-RED. Eles certamente possuem mecanismos de segurança, porém nenhum teste foi feito. Os riscos de permitir a atuação remoto sobre cargas importantes requer que um estudo mais aprofundado neste tema antes que o sistema seja implementado plenamente.

6.2 Trabalhos Futuros

O sistema de monitoramento ainda pode ser expandido e com a inclusão do servidor Node-RED várias novas aplicações podem ser exploradas. Algumas sugestões para trabalhos futuros são detalhadas a seguir.

6.2.1 Expansão dos nós medidores

Nós medidores de outras grandezas, como vazão de água e temperatura, poderiam ser integrados ao sistema existentes. Através do servidor Node-RED diferentes dispositivos e protocolos podem ser utilizados caso sejam mais apropriados. A inclusões de múltiplos nós também pode ser implementado para estudar o sistema supervisório em condições de alta carga.

6.2.2 Estudo da segurança do sistema supervisório

Como mencionado anteriormente, as opções de segurança oferecidas pelo protocolo MQTT precisão ser estudadas e testadas antes que ele possa ser utilizado para enviar as mensagens vitais trocadas entre os nós e o servidor. A segurança também deve ser analisada no lado do servidor Node-RED, onde a interface com o usuário necessitária de autenticação e o reencaminhamento de mensagens ao EmonCMS deve levar em consideração sua chave de API.

6.2.3 Desenvolvimento de uma interface unificada para o sistema supervisório

O desenvolvimento de uma interface para as configurações das aplicações do Node-RED a fim de facilitar o seu uso poderia ser criada e integrada à interface do servidor EmonCMS existente. Essa interface poderia ser construída através da programação em fluxo do Node-RED utilizando de nós de entrada e resposta HTTP.

6.2.4 Estudo de métodos de tolerância a falhas para o sistema supervisor

Para prover tolerância a falhas nos servidores um esquema de redundância seria interessante. Isso poderia ser feito através do *backup* dos fluxos do Node-RED para outro servidor e um *bridge* entre mediadores MQTT.

Referências

ARDUINO. *Arduino - Introduction*. 2015. Página de introdução ao Arduino. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 29 de junho de 2015. Citado na página 27.

GUINARD, D.; TRIFA, V.; WILDE, E. A resource oriented architecture for the web of things. In: *Internet of Things (IOT), 2010*. [S.l.: s.n.], 2010. p. 1–8. Citado na página 32.

HUDSON, G.; LEA, T. *OpenEnergyMonitor*. 2015. Homepage do OpenEnergyMonitor. Disponível em: <http://openenergymonitor.org/emon/>. Acesso em: 27 de abril de 2015. Citado 6 vezes nas páginas 9, 21, 22, 23, 61 e 69.

JSON. *JSON*. 2015. Página de introdução ao JSON. Disponível em: <http://json.org/>. Acesso em: 30 de junho de 2015. Citado na página 26.

JúNIOR, P. A. da S. et al. *Desenvolvimento e implantação de um sistema supervisório de energia elétrica no IFSC-SJ*. 2014. Disponível em: <http://eventoscientificos.ifsc.edu.br/index.php/sepei/sepei2014/paper/view/666>. Acesso em: 27 de abril de 2015. Citado na página 19.

LAMPKIN, V. et al. *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*. 2012. Redbook da IBM sobre MQTT. Disponível em: <http://www.redbooks.ibm.com/redbooks/pdfs/sg248054.pdf>. Acesso em: 13 de fevereiro de 2016. Citado 5 vezes nas páginas 9, 11, 19, 35 e 37.

LEA, R.; BLACKSTOCK, M.; CALDERON, R. *Node-RED: Lecture 1 – A brief introduction to Node-RED*. 2016. Aula de introdução ao MQTT. Disponível em: <http://noderedguide.com/index.php/2015/10/01/nr-lecture-1/>. Acesso em: 26 de março de 2016. Citado 2 vezes nas páginas 9 e 34.

LEA, T. *emoncms*. 2015. Homepage do emoncms. Disponível em: <http://emoncms.org/>. Acesso em: 27 de abril de 2015. Citado na página 24.

LOBUNETS, O.; KRYLOVSKIY, A. *Applying Flow-based Programming Methodology to Data-driven Applications Development for Smart Environments*. 2014. Artigo sobre a aplicação da programação baseada em fluxo em aplicativos para ambientes inteligentes. Disponível em: <https://www.thinkmind.org/download.php?articleid=ubicomm_2014_8_10_10134>. Acesso em: 15 de fevereiro de 2016. Citado na página 32.

MYSQL. *MySQL ___ MySQL 5.6 Reference Manual ___ 1 General Information*. 2015. Página de introdução ao MySQL. Disponível em: <http://dev.mysql.com/doc/refman/5.6/en/introduction.html>. Acesso em: 30 de junho de 2015. Citado na página 26.

OASIS. *MQTT*. 2015. MQTT Padrão OASIS. Disponível em: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>. Acesso em: 23 de outubro de 2015. Citado na página 35.

O'LEARY, N. *PubSubClient.h*. 2016. Disponível em: <https://github.com/knolleary/pubsubclient>. Citado na página 48.

O'LEARY, N.; CONWAY-JONES, D. *Node-RED*. 2016. Disponível em: <https://github.com/node-red/node-red>. Citado na página 34.

OPENSTACK. *Software » OpenStack Open Source Cloud Computing Software*. 2015. Página de introdução ao OpenStack. Disponível em: <http://www.openstack.org/software/>. Acesso em: 29 de junho de 2015. Citado 2 vezes nas páginas 9 e 27.

PHP-HTML. *Model View Controller(MVC) in PHP*. 2015. Tutorial do MVC. Disponível em: <http://php-html.net/tutorials/model-view-controller-in-php/>. Acesso em: 27 de abril de 2015. Citado 2 vezes nas páginas 9 e 25.

REDIS. *An introduction to Redis data types and abstractions – Redis*. 2015. Pagina de introdução aos tipos de dados do Redis. Disponível em: <http://redis.io/>. Acesso em: 30 de junho de 2015. Citado na página 26.

TORRESINI, E. *OpenStack IFSC - IF-SC São José*. 2015. Descrição do sistema OpenStack do campus IFSC-SJ. Disponível em: <http://wiki.sj.ifsc.edu.br/wiki/index.php/OpenStack_IFSC>. Acesso em: 29 de junho de 2015. Citado 2 vezes nas páginas 9 e 28.

WENDE, J. *Node-RED A Visual Tool for building the Internet of Things*. 2016. Slides de apresentação do Node-RED. Disponível em: <http://www.iot-vienna.at/global-iot-day-event/2015/lib/exe/fetch.php?media=talks:wende_joerg:gide2015_node-red.pdf>. Acesso em: 26 de março de 2016. Citado 2 vezes nas páginas 9 e 33.

Y.2060, I.-T. *Recommendation ITU-T Y.2060 Overview of the Internet of things*. [S.l.], 2012. Citado na página 31.

Anexos

ANEXO A – Código do Arduino da Subestação

Na lista A.1 a seguir é listado o código de envio das medições do Arduino que está na subestação.

A equação $Potência = ((Medição - 180)/900.24) * 132000$ é utilizada para o calculo da potência conforme a linha 92.

Listagem A.1 – Código do Arduino instalado na subestação.

```
1  char foo;
2  //#define WIFI
3
4  #include <SPI.h>
5
6  int potpin = 0;
7  float val = 0;
8  float poten;
9  //nome do input no emoncms e futuramente do topico MQTT
10 char nodename[] = "Potencia1";
11
12 #include <Ethernet.h>
13 // Include Emon Library
14 #include "EmonLib.h"
15
16 //network configuration WIRED
17   byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x69, 0xD5};
18
19   IPAddress ip(172, 168, 1, 2);
20   IPAddress subnet(255, 255, 255, 0);
21   IPAddress DNS(8, 8, 8, 8);
22   IPAddress gw(172, 168, 1, 254);
23
24   EthernetClient client;
25
26 float t = 0;
27
28 // Create an Emon instance
29 EnergyMonitor emon1;
30
31 //Emoncms configurations
32 //char server[] = "emoncms.org";      // name address for emoncms.org
33 IPAddress server(200, 135, 233, 25);  // numeric IP for emoncms.org (no DNS
```

```
    )
34
35 String apikey = "2b906d9bce32c46223359b53fc27e050";  //api key
36 int node = 1; //if 0, not used
37
38 unsigned long lastConnectionTime = 0;           // last time you connected
    to the server, in milliseconds
39 boolean lastConnected = false;                 // state of the connection
    last time through the main loop
40 const unsigned long postingInterval = 5*1000;  // delay between updates, in
     milliseconds
41
42 void setup() {
43   // start serial port:
44   Serial.begin(9600);
45
46   // Display a welcome message
47   Serial.println("Emoncms client starting...");
48
49   if (!Ethernet.begin(mac)) {
50     // if DHCP fails, start with a hard-coded address:
51     Serial.println("Failed to get an IP address using DHCP, forcing
        manually");
52     Ethernet.begin(mac, ip, dns, gw, subnet);
53   }
54
55   printStatus();
56 }
57
58
59
60 void loop() {
61
62   // if there's incoming data from the net connection.
63   // send it out the serial port.  This is for debugging
64   // purposes only:
65   if (client.available()) {
66     char c = client.read();
67     Serial.print(c);
68   }
69
70   // if there's no net connection, but there was one last time
71   // through the loop, then stop the client:
72   if (!client.connected() && lastConnected) {
73     Serial.println();
74     Serial.println("Disconnecting...");
75     client.stop();
```

```
76    }
77
78    // if you're not connected, and at least <postingInterval> milliseconds
          have
79    // passed sinceyour last connection, then connect again and
80    // send data:
81    if(!client.connected() && (millis() - lastConnectionTime >
          postingInterval)) {
82
83    // Led inizio ciclo
84    digitalWrite(2, HIGH);
85
86
87
88
89
90    //send values
91  val = analogRead(potpin);
92  poten=(((val-180)/900.24)*132000);
93  if(poten<0){
94  poten=0;
95  }
96    sendData();
97
98
99   }
100   // store the state of the connection for next time through
101   // the loop:
102   lastConnected = client.connected();
103 }
104
105 // this method makes a HTTP connection to the server:
106 void sendData() {
107   // if there's a successful connection:
108   if (client.connect(server, 80)) {
109     Serial.println("Connecting...");
110     // send the HTTP GET request:
111     client.print("GET /nodered/input/post?apikey=");
112     client.print(apikey);
113     if (node > 0) {
114       client.print("&node=");
115       client.print(node);
116     }
117     client.print("&json?node=1&json={");
118     client.print(nodename);
119     client.print(":");
120     client.print(poten);
```

```
121       client.print("}");
122       client.println("} HTTP/1.1");
123       client.println("Host:emoncms.org");
124       client.println("User-Agent: Arduino-ethernet");
125       client.println("Connection: close");
126       client.println();
127
128
129       // note the time that the connection was made:
130       lastConnectionTime = millis();
131     }
132     else {
133       // if you couldn't make a connection:
134       Serial.println("Connection failed");
135       Serial.println("Disconnecting...");
136       client.stop();
137     }
138
139     // led fine ciclo
140     digitalWrite(2, LOW);
141 }
142 void printStatus() {
143       // print your local IP address:
144       Serial.print("IP address: ");
145       for (byte thisByte = 0; thisByte < 4; thisByte++) {
146         // print the value of each byte of the IP address:
147         Serial.print(Ethernet.localIP()[thisByte], DEC);
148         Serial.print(".");
149       }
150     Serial.println();
151
152 }
```

ANEXO B – Gerador de Relatórios

O modulo gerador de relatórios pode ser encontrado no canto superior esquerdo da interface do EmonCMS, como mostrado na figura 23. Ele requer apenas que o usuário escolha o *feed* e datas inciais e finais do período com os dados desejados. O arquivo resultante usa o formato *Comma-separated values* (CSV), que pode ser visualizado em um programa de edição de tabelas como o Excel.

Figura 23: Interface do gerador de relatórios.(HUDSON; LEA, 2015)

O código dos arquivos *view*, *model* e *controller* são paresentados nas listas B.1, B.2 e B.3 respectivamente.

Listagem B.1 – Código PHP do *view* gerador de relatorio.

```
1 <?php
2     global $path;
3 ?>
4 <script type="text/javascript" src="<?php echo $path; ?>Modules/user/user.
    js"></script>
5 <script type="text/javascript" src="<?php echo $path; ?>Modules/feed/feed.
    js"></script>
6 <script type="text/javascript" src="<?php echo $path; ?>Modules/relatorio/
    relatorio.js"></script>
7 <script type="text/javascript" src="<?php echo $path; ?>Lib/tablejs/table.
    js"></script>
8 <script type="text/javascript" src="<?php echo $path; ?>Lib/tablejs/custom-
    table-fields.js"></script>
9 <link href="<?php echo $path; ?>Lib/bootstrap-datetimepicker-0.0.11/css/
    bootstrap-datetimepicker.min.css" rel="stylesheet">
```

```
10 <script type="text/javascript" src="<?php echo $path; ?>Lib/bootstrap-
       datetimepicker-0.0.11/js/bootstrap-datetimepicker.min.js"></script>
11
12
13 <style>
14 #table input[type="text"] {
15          width: 88%;
16 }
17 </style>
18
19 <br>
20
21     <h3 id="ExportModalLabel">Relatorio</h3>
22     <div class="modal-body">
23     <p><b><span id="SelectedExportFeed"></span></b></p>
24     <p>Selecione o feed e o intervalo de tempo desejado: </p>
25
26         <table class="table">
27         <tr>
28             <td>
29                 <p><b>Feed</b></p>
30                 <select id="feedbox" required="required">
31                 <option value="">Selecione um Feed</option>
32                 </select>
33             </td>
34             <td>
35                 <p><b>Data Inicial</b></p>
36                     <div id="datetimepicker1" class="input-append date">
37                     <input id="export-start" data-format="dd/MM/yyyy hh:mm:
                           ss" type="text" />
38                     <span class="add-on"> <i data-time-icon="icon-time"
                           data-date-icon="icon-calendar"></i></span>
39                     </div>
40             </td>
41             <td>
42                 <p><b>Data Final</b></p>
43                 <div id="datetimepicker2" class="input-append date">
44                     <input id="export-end" data-format="dd/MM/yyyy hh:mm:ss
                           " type="text" />
45                     <span class="add-on"> <i data-time-icon="icon-time"
                           data-date-icon="icon-calendar"></i></span>
46                 </div>
47             </td>
48         </tr>
49             <td><br><button class="btn" id="export">Exportar</button></td><
                   td><br>Tamanho estimado do arquivo: <span id="downloadsize"
                   >0</span>kB</td>
```

```
50          </tr>
51          </table>
52      </div>
53  </div>
54
55  <script>
56
57      var apikeystr = "?apikey=2b906d9bce32c46223359b53fc27e050";
58      var path = "<?php echo $path; ?>";
59
60      console.log(path+"feed/list.json"+apikeystr);
61      $.ajax({ url: path+"feed/list.json"+apikeystr, dataType: 'json', async:
            true, success: function(data, textStatus, xhr) {
62      if ( console && console.log ) {
63        caixa = data;
64      select = document.getElementById('feedbox');
65      for (z in caixa){
66          var opt = document.createElement('option');
67          opt.value = caixa[z]['id'];
68          opt.text = caixa[z]['name'];
69          select.appendChild(opt);
70      }
71      }}});
72
73      $('#datetimepicker1').datetimepicker({
74          language: 'pt-BR.utf8'
75      });
76
77      $('#datetimepicker2').datetimepicker({
78          language: 'pt-Br.utf8'
79      });
80
81      $('#datetimepicker1, #datetimepicker2').on('changeDate', function(e)
82      {
83          var export_start = parse_timepicker_time($("#export-start").val());
84          var export_end = parse_timepicker_time($("#export-end").val());
85          var export_interval = '300';
86          var downloadsize = (((export_end - export_start) / 1000) /
                export_interval) * 17; // 17 bytes per dp
87          $("#downloadsize").html((downloadsize/1024).toFixed(0));
88      });
89
90      $("#export").click(function()
91      {
92          var feedid = $("#feedbox").val();
93          var export_start = parse_timepicker_time($("#export-start").val());
94          var export_end = parse_timepicker_time($("#export-end").val());
```

```
95          var export_interval = '300';
96          var export_timezone_offset = '';
97
98          if (feedid=="") {alert("Selecione um Feed"); return false; }
99          if (!export_start) {alert("Please enter a valid start date");
                return false; }
100         if (!export_end) {alert("Please enter a valid end date"); return
                false; }
101         if (export_start>=export_end) {alert("Start date must be further
                back in time than end date"); return false; }
102         if (export_interval=="") {alert("Please select interval to download
                "); return false; }
103         var downloadsize = ((export_end - export_start) / export_interval)
                * 17 / 1000; // 17 bytes per dp
104
105         if (downloadsize>(10*1048576)) {alert("Download file size to large
                (download limit: 10Mb)"); return false; }
106
107         window.open(path+"relatorio/tabela.json?id="+feedid+"&start="+(
                export_start+(export_timezone_offset))+"&end="+(export_end+(
                export_timezone_offset))+"&interval="+export_interval);
108     });
109
110     function parse_timepicker_time(timestr)
111     {
112         var tmp = timestr.split(" ");
113         if (tmp.length!=2) return false;
114
115         var date = tmp[0].split("/");
116         if (date.length!=3) return false;
117
118         var time = tmp[1].split(":");
119         if (time.length!=3) return false;
120
121         return new Date(date[2],date[1]-1,date[0],time[0],time[1],time
                [2],0).getTime();
122     }
123
124 </script>
```

Listagem B.2 – Código PHP do *model* gerador de relatorio.

```
1  <?php
2
3  class Relatorio
4  {
5
6      public function tabela($id,$start,$end,$interval)
```

```
7       {
8
9           global $path, $mysqli, $redis, $feed_settings;
10
11          include "Modules/feed/feed_model.php";
12          $feed = new Feed($mysqli,$redis,$feed_settings);
13
14          $csvtable = $feed->get_data($id,$start,$end,$interval,'1','1');
15
16          // There is no need for the browser to cache the output
17          header("Cache-Control: no-cache, no-store, must-revalidate");
18
19          // Tell the browser to handle output as a csv file to be downloaded
20          header('Content-Description: File Transfer');
21          header("Content-type: application/octet-stream");
22          $filename = $id.".csv";
23          header("Content-Disposition: attachment; filename={$filename}");
24
25          header("Expires: 0");
26          header("Pragma: no-cache");
27
28          $exportfh = @fopen( 'php://output', 'w' );
29
30          $i = 0;
31          $dp = count($csvtable);
32          fwrite($exportfh, "Data;Hora;Potencia\n");
33
34          while ($i < $dp)
35          {
36              fwrite($exportfh, strftime("%d/%m/%y;%R", $csvtable[$i][0]
                    /1000).";".$csvtable[$i][1]."\n");
37              $i++;
38          }
39
40          fclose($exportfh);
41          exit;
42
43      }
44
45      public function lista_feedold()
46      {
47          $mysqli = new mysqli("localhost", "root", "W94q608X", "emoncms");
48
49          if (mysqli_connect_errno()) {
50              printf("Conexao falhou: %s\n", mysqli_connect_error());
51              exit();
52          }
```

```
53
54          $query = "SELECT id,name,tag FROM feeds";
55          $resultado = $mysqli->query($query);
56
57          while ($row = mysqli_fetch_array($resultado, MYSQLI_ASSOC)) {
58          printf("%s - %s - %s\n", $row["id"], $row["name"], $row["tag"]);
59          }
60
61          $resultado->close();
62          $mysqli->close();
63      }
64
65      public function list_feed()
66      {
67
68          $result = $this->mysqli->query("SELECT id,name,tag FROM feeds");
69          $feeds = array();
70          while ($row = $result->fetch_object()) $feeds[] = $row;
71          return $feeds;
72      }
73
74  }
75
76  ?>
```

Listagem B.3 – Código PHP do *controller* gerador de relatorio.

```
1   <?php
2
3     // no direct access
4     defined('EMONCMS_EXEC') or die('Restricted access');
5
6     function relatorio_controller()
7     {
8       global $session,$route;
9       $result = false;
10
11      include "Modules/relatorio/relatorio_model.php";
12      $relatorio = new Relatorio();
13
14      if ($route->format == 'html')
15      {
16
17          if ($route->action == "view") $result = view("Modules/relatorio/
                relatorio_view.php",array());
18
19      }
20
```

```
21      if ($route->format == 'json')
22      {
23          if ($route->action == "tabela") $result = $relatorio->tabela(get('
                id'),get('start'),get('end'),get('interval'));
24          if ($route->action == "listafeed") $result = $relatorio->lista_feed
                ();
25      }
26
27      return array('content'=>$result);
28  }
```

ANEXO C – Esquemático do emonTx V3

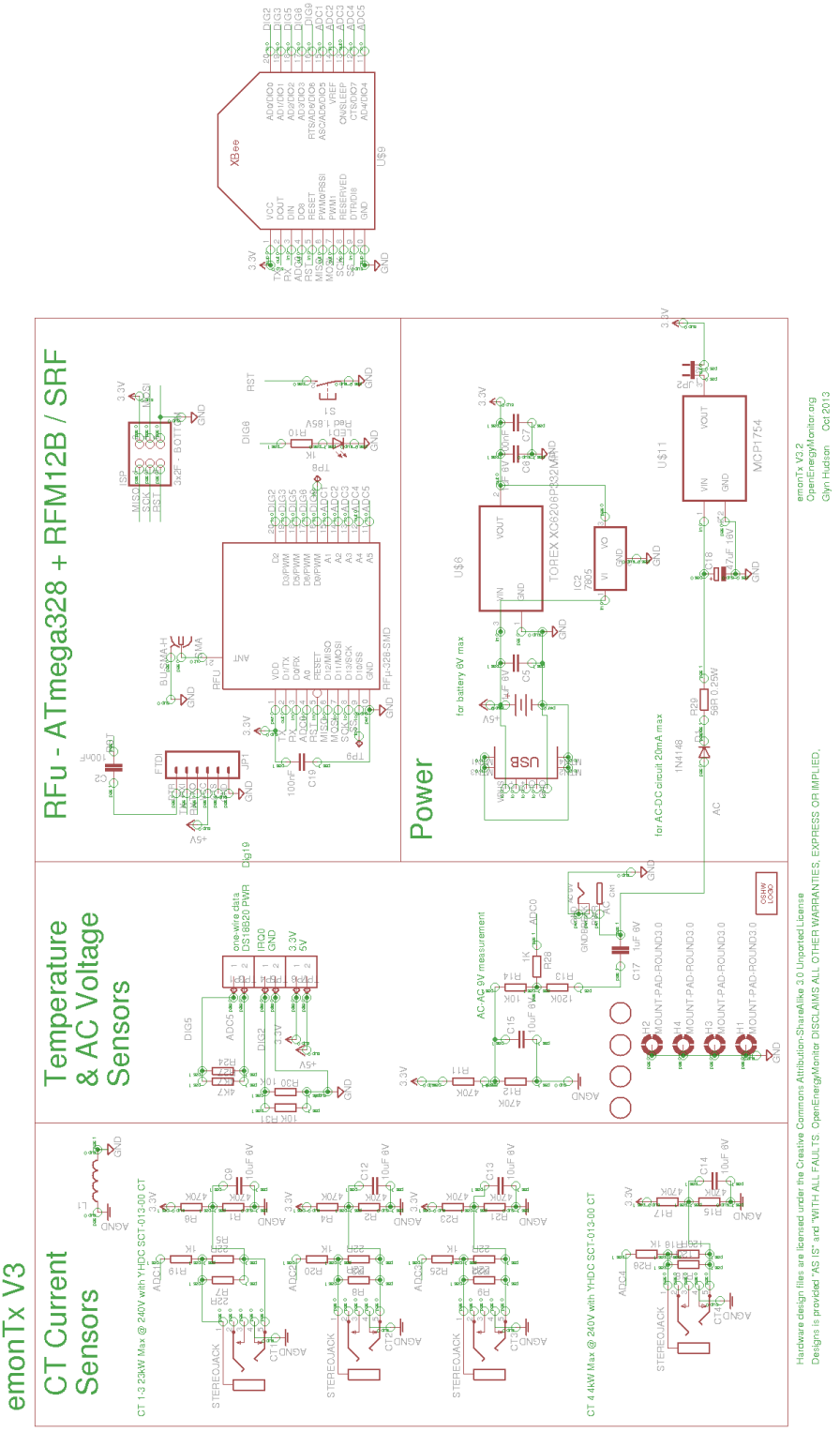

Figura 24: Esquemático do emonTx V3.(HUDSON; LEA, 2015)