UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETROTÉCNICA
CURSO DE ENGENHARIA INDUSTRIAL ELÉTRICA / AUTOMAÇÃO

FABIANNA STUMPF TONIN
GUILHERME FRANCESCON CITTOLIN
VINICIUS DE SOUZA

DESENVOLVIMENTO DE UM SISTEMA WEB DE CONTROLE DE ACESSO

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2015

FABIANNA STUMPF TONIN
GUILHERME FRANCESCON CITTOLIN
VINICIUS DE SOUZA

DESENVOLVIMENTO DE UM SISTEMA WEB DE CONTROLE DE ACESSO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Engenharia Industrial Elétrica com ênfase em Automação do Departamento Acadêmico de Eletrotécnica – DAELT – da Universidade Tecnológica Federal do Paraná – UTFPR.

Orientador: Prof. Dr. Winderson Eugenio dos Santos

Co-Orientador: Prof. Dr. Marcelo de Oliveira Rosa

CURITIBA
2015

Fabianna Stumpf Tonin
Guilherme Francescon Cittolin
Vinicius de Souza

Desenvolvimento de um Sistema Web de Controle de Acesso

Este Trabalho de Conclusão de Curso de Graduação foi julgado e aprovado como requisito parcial para a obtenção do Título de Engenheiro Eletricista do curso de Engenharia Industrial Elétrica - ênfase Automação do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR).

Curitiba, 09 de Fevereiro de 2015.

_____
Prof. Paulo Sérgio Walenia, Esp.
Coordenador de Curso
Engenharia de Controle e Automação

_____
Prof. Marcelo de Oliveira Rosa, Dr.
Responsável pelos Trabalhos de Conclusão de Curso
Engenharia de Controle e Automação do DAELT

ORIENTAÇÃO

_____
Winderson Eugênio dos Santos, Dr.
Universidade Tecnológica Federal do Paraná
Orientador

_____
Marcelo de Oliveira Rosa, Dr.
Universidade Tecnológica Federal do Paraná
Co - Orientador

BANCA EXAMINADORA

_____
Alceu André Badin, Dr.
Universidade Tecnológica Federal do Paraná

_____
Marcelo de Oliveira Rosa, Dr.
Universidade Tecnológica Federal do Paraná

_____
Marco José da Silva, PhD.
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia Industrial Elétrica – ênfase Automação

AGRADECIMENTOS

Agradecemos a Deus por estar sempre ao nosso lado, dando-nos forças para que nenhum obstáculo se mostrasse intransponível e conseguíssemos concluir mais essa etapa;

Nosso eterno e sincero agradecimento aos nossos pais pelo apoio, amor e carinho, por lutarem pela nossa educação e nos privilegiarem com bons estudos. Obrigado pela compreensão nos momentos em que a dedicação aos estudos foi exclusiva;

Aos nossos familiares, amigos e namorados nosso agradecimento especial. Sem a presença e apoio de vocês, essa trajetória não teria sido tão prazerosa;

Aos nossos orientadores Winderson Eugenio dos Santos e Marcelo de Oliveira Rosa, agradecemos pelos ensinamentos e pela paciência que contribuíram de forma especial para a conclusão dessa monografia. Agradecemos também ao Peterson Senha pelo auxílio no início do projeto;

Nosso agradecimento especial ao Mauro e Osmar da OMS Engenharia que acreditaram e financiaram o nosso projeto. Somos gratos pelo apoio e conselhos dispensados na concretização desse projeto;

Por fim, o nosso muito obrigado a todos que contribuíram direta ou indiretamente para que esse trabalho fosse realizado, os nossos sinceros AGRADECIMENTOS.

RESUMO

TONIN Fabianna S.; CITTOLIN, Guilherme F.; SOUZA, Vinicius de. Desenvolvimento de um Sistema Web de Controle de Acesso. 2015. 69f. Trabalho de Conclusão de Curso (Engenharia Industrial Elétrica – ênfase Automação) – Departamento Acadêmico de Eletrotécnica – Universidade Tecnológica Federal do Paraná - Curitiba, 2015.

A Internet, juntamente com os sistemas e serviços nela presentes, tem se mostrado cada vez mais presente em todas as atividades do dia-a-dia. Portanto, atualmente existe um aumento dos esforços para desenvolvimento das utilidades desta plataforma, o que leva a um desenvolvimento de sistemas de informação baseados na tecnologia. A integração dos sistemas de automação com a tecnologia da Internet demanda dispositivos caros e complexos, o que torna esse processo inflexível e caro. Portanto, desenvolveu-se, neste trabalho, um sistema de controle de acesso integrado à Internet de baixo custo, o qual permite a autorização, o monitoramento e o registro da entrada e saída de pessoas de ambientes, como salas e laboratórios, por exemplo.

Palavras-chave: Internet. Controle. Segurança.

ABSTRACT

TONIN Fabianna S.; CITTOLIN, Guilherme F.; SOUZA, Vinicius de. Development of a Web-based System for Access Control. 2015. 69f. Trabalho de Conclusão de Curso (Engenharia Industrial Elétrica – ênfase Automação) – Departamento Acadêmico de Eletrotécnica – Universidade Tecnológica Federal do Paraná - Curitiba, 2015.

The internet, with the systems and services offered, has been even more present in all daily activities. Therefore, there is nowadays an increasing effort to develop the utilities of this platform, which favors the development of the information systems based on it. Although, the integration of automation systems with the internet still demands complex and expensive devices, making the whole process neither worthy nor flexible. This work develops a low-cost approach of an access control system integrated to the internet, that allows the authorization, monitoring and registry of entrance and exit of people from specified rooms, like classrooms and laboratories.

Keywords: Internet. Control. Security.

LISTA DE FIGURAS

LISTA DE TABELAS

ÍNDICE DE SIGLAS

API	*Application Programming Interface*
AMQP	*Advance Message Queuing Protocol*
CI	Circuito Integrado
DAELT	Departamento Acadêmico de Eletrotécnica
FTP	*File Transfer Protocol*
GPIO	*General Purpose Input/Output*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*
J2EE	Java 2 *Enterprise Edition*
PC	*Personal Computer*
SCADA	Supervisory Control and Data Acquisition
SMD	*Surface Mounted Device*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
UML	*Unified Modeling Language*
USB	*Universal Serial Bus*
UTFPR	Universidade Tecnológica Federal do Paraná
WIS	*Web-based Information Systems*

CONTEÚDO

1 INTRODUÇÃO

1.1 DELIMITAÇÃO DO TEMA

O presente trabalho trata do desenvolvimento de um sistema Web de controle de acesso. Sistemas de informação baseados na tecnologia Web (WIS, *Web-based Information System*) têm sido desenvolvidos com o objetivo de reduzir a necessidade de trabalho humano na produção de bens e serviços. Estes sistemas WISs suportam trabalho integrado com outros sistemas não baseados em Web, como bancos de dados e sistemas de processamento de transações. WISs podem integrar processos ou sistemas em uma única interface e, também, permitir o acesso de uma intranet local ou da rede global de Internet.

Chou (2008) considera a rede mundial de Internet uma importante ferramenta, pois pode: gerar requisições dinamicamente para usuários finais; permitir que o sistema pode ser acessado de qualquer lugar com acesso à Internet; possibilitar a atualização e melhoramento do sistema com a adição de módulos ou funções; e, com a ajuda de aplicações de *software* abertas, o programa todo pode ser desenvolvido com baixo custo financeiro. Segundo Kirubashankar et al. (2011), tem havido um constante aumento no desenvolvimento da automação industrial através de monitoramento remoto e diagnóstico virtual. A automação baseada na Internet é possível com a utilização de elementos que se comunicam com a rede, como os sistemas de supervisão e aquisição de dados SCADA. Kirubashankar et al. também citam como objetivos do monitoramento remoto a prevenção de indisponibilidade não planejada do sistema e a otimização do controle do sistema.

A integração do controle de acesso a um sistema Web traz diversas funcionalidades ao sistema, como a possibilidade de controle e monitoramento remoto, a possibilidade de acesso à informação de qualquer lugar e segurança da informação.

> No mundo contemporâneo, quando se fala em segurança nas empresas é muito comum a associação das áreas de segurança patrimonial e tecnologia da informação e comunicação (TIC), para a elaboração e implantação de um projeto de controle de acesso. (THOMÉ et al., p. 2).

Segundo Peixoto (2013, p. 8), a utilização de sistemas de controle de acesso vem aumentando em instituições e ambientes organizacionais. Há uma necessidade

de maior segurança para usuários e instituições, abrindo espaço para este tipo de tecnologia. Ainda há empecilhos financeiros e tecnológicos para a instalação destes sistemas, já que “ainda possuem alto custo de implantação e geralmente não possuem padronização das funcionalidades, principalmente em se tratando do sistema que os gerenciam” (PEIXOTO, 2013, p. 8).

A Universidade Tecnológica Federal do Paraná possui laboratórios com equipamentos e computadores, onde há a necessidade de monitoramento e controle das pessoas que entram e saem do laboratório. Atualmente, utilizam-se chaves nas portas dos laboratórios, caracterizando-se um sistema inseguro de controle de acesso, visto que podem ser copiadas, perdendo-se o controle das pessoas que podem acessar os laboratórios. “Os requisitos de segurança (...) são essenciais a um sistema de controle de acesso. Por possuírem constantes investimentos em tecnologia e equipamentos, as universidades públicas são muito visadas por criminosos.” (PEIXOTO, 2013, p. 16).

Um sistema de controle de acesso automático baseado em autenticação do usuário e registro das entradas e saídas dos laboratórios é capaz de tornar o ambiente mais seguro. Além disso, traz mais flexibilidade e acessibilidade de forma segura ao laboratório, visto que a pessoa autorizada pode entrar e sair conforme definição e prévia configuração, mantendo em registro todas as ações.

1.2 PROBLEMAS E PREMISSAS

De acordo com o engenheiro José Roberto Muratori, presidente da Associação Brasileira de Automação Residencial (Aureside), o crescente uso de equipamentos capazes de se conectar à Internet e a integração de sistemas via circuitos informatizados e rede elétrica contribuem para o aumento do interesse em desenvolver soluções integradas à rede mundial de computadores, que são capazes de proporcionar maior facilidade e conforto para os usuários. (MARQUES, 2014)

Contudo, o problema reside no fato de que essas tecnologias possuem preços relativamente altos e são de difícil implementação. (CRUZ, 2009). Uma aplicação desta tecnologia é o controle de acesso de pessoas em ambientes classificados, como por exemplo, a catraca 4660008ATZ200 do fabricante CFTV Center, que custa mais de R$ 9000,00 (CFTV Center, 2015).

A possibilidade de integração através da Web permitiria que todos os avanços e recursos da Tecnologia de Informação (TI) tornassem tais sistemas menos onerosos em relação ao monitoramento e manutenção dos mesmos, se comparado com os atuais sistemas de controle de acesso existentes no mercado.

Portanto, a principal premissa deste trabalho está relacionada à execução de um projeto de baixo custo, utilizando os atuais recursos de Tecnologia da Informação, destinados à aplicação em um controle de acesso em laboratórios didáticos.

1.3 OBJETIVOS

1.3.1 OBJETIVO GERAL

O principal objetivo desse trabalho é desenvolver um sistema de controle de acesso integrado à Web, que permita a autorização, o monitoramento e o registro da entrada e saída de pessoas em ambientes de uso restrito, especificamente: salas de professores e laboratórios didáticos.

1.3.2 OBJETIVOS ESPECÍFICOS

- Revisar a teoria inerente aos principais conceitos de monitoramento e controle de sistemas de acesso de edificações.
- Revisar a teoria relacionada a métodos de modelagem de sistemas automatizados.
- Projetar um sistema robusto à indisponibilidade, tanto conectiva como energética.
- Tornar o sistema dinâmico às modificações futuras, ou seja, desenvolver um sistema flexível.
- Relacionar os principais recursos de Tecnologia da Informação disponíveis para um desenvolvimento de baixo custo da aplicação.
- Implementar o sistema de controle de acesso em ambientes selecionados do Departamento Acadêmico de Eletrotécnica do Campus Curitiba da Universidade Tecnológica Federal do Paraná - UTFPR.

1.4 JUSTIFICATIVA

De acordo com Marques (2014), a automação é um reflexo das mudanças impostas à vida contemporânea, na qual, segurança, conforto e praticidade tornaram-se necessidades básicas. Para tanto, torna-se evidente a necessidade dos ambientes residenciais e comerciais serem adequados às novas tendências tecnológicas. Tendo por base tal premissa de garantia de segurança e qualidade de vida, o trabalho em questão desenvolve e implementa um modelo para controle de acesso confiável. Além disso, este projeto apresenta característica de permitir o acesso global, via Internet, por parte do usuário, assegurando assim, maior conforto. A peculiaridade do acesso à configuração do sistema via Internet não é ofertada pelos dispositivos existentes no mercado.

1.5 ESTRUTURA DO TRABALHO

O trabalho é dividido em cinco capítulos. O primeiro apresenta uma introdução geral seguida de uma descrição do problema, sua justificativa, os objetivos tanto gerais como específicos. O segundo capítulo é destinado à revisão bibliográfica, a qual constitui o embasamento teórico necessário para o desenvolvimento do projeto. Por sua vez, no terceiro capítulo realiza-se a descrição da parte experimental, e a especificação do projeto. No capítulo quatro detalham-se os resultados de testes realizados. Por fim, no capítulo cinco são apresentadas as conclusões, os comentários finais e as sugestões para os próximos trabalhos.

2 REFERENCIAL TEÓRICO

Neste capítulo é exposta uma revisão da literatura sobre os principais tópicos, conceitos e tecnologias que serviram como base para o desenvolvimento deste trabalho.

2.1 COMPUTADORES EMBARCADOS

Computadores embarcados são sistemas de *hardware* microprocessados, construídos para desempenhar uma função ou conjunto de funções específicos. (HEATH, 2003). Exemplos destes Computadores embarcados encontrados no mercado são nomeadamente: o Arduino, o Android PC System, e o Raspberry Pi, este último abordado em mais detalhes na seção abaixo.

2.1.1 RASPBERRY PI

O Raspberry Pi é um microcomputador compatível com sistemas operacionais baseados na arquitetura ARMv6, e portanto, qualquer linguagem que possa ser compilada nessa arquitetura pode ser usada para o desenvolvimento de *softwares*. Em virtude disso, o Raspberry Pi pode ser aplicado a inúmeras finalidades a citar: projetos de eletrônica, reprodução de vídeos de alta definição e diversas atividades que o computador convencional executa como planilhas, processamento de textos e jogos. (RASPEBERRY PI FOUNDATION, 2014)

Esse computador do tamanho de um cartão de crédito foi desenvolvido pela Fundação Raspberry Pi, sediada no Reino Unido, com o intuito de ensinar e incentivar as crianças a programarem. E, em função desse propósito, tal dispositivo possui baixos custos para ser adquirido. A Figura 1 mostra todos os componentes presentes no *hardware* desse microcomputador. (RASPBERRY PI FOUNDATION, 2014).

Quando da pesquisa, o custo desse microcomputador girava em torno de 35 dólares mais impostos. (ELEMENT 14, 2014)

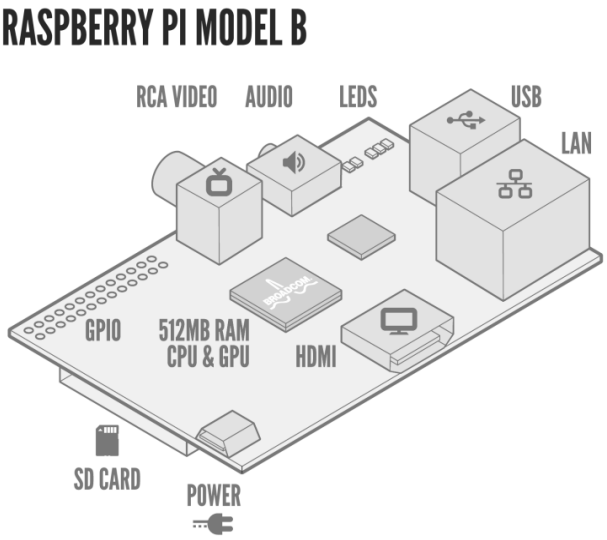

Figura 1 - Organização dos componentes em um Raspberry Pi
(RASPBERRY PI, 2014)

2.2 LINGUAGENS DE PROGRAMAÇÃO COM SUPORTE À WEB

Como os computadores conseguem apenas interpretar e executar códigos de máquina foram desenvolvidas as várias linguagens de programação existentes, que se diferem na sintaxe e nos recursos oferecidos. Para tradução das instruções para linguagem de máquina, cada linguagem possui um compilador ou interpretador, que é o responsável por transformar os comandos listados nestes códigos binários que, enfim, são compreendidas pelo processador (MORIMOTO, 2011).

Algumas destas linguagens oferecem bibliotecas de código com recursos específicos para a comunicação em rede, tornando o desenvolvimento mais rápido e menos suscetível a erros. Exemplos dessas linguagens são Java e Javascript, as quais estão descritas abaixo.

2.2.1 JAVA

Java é uma linguagem de programação desenvolvida pela empresa Sun Microsystems na década de 90, orientada a objeto (modelada em torno de objetos ao invés de ações), e foi criada tendo em vista os seguintes princípios: simplicidade, familiaridade, robustez e segurança. Ademais, o Java é distribuído gratuitamente na Internet. (CADENHEAD, 2005)

Ainda, quando desenvolvida, esta linguagem seguiu o lema "*write once, run anywhere"*, ou seja, "escreva uma vez, execute em qualquer lugar". Deste modo, os programas em Java são independentes de plataforma, ou seja, eles podem ser executados em qualquer sistema que possua uma máquina virtual Java (JVM – *Java Virtual Machine*). JVM é um programa com função de carregar e executar os aplicativos Java, convertendo-os para *bytecodes* (forma intermediária de código) os quais são, em seguida, interpretados. Isto permite que aplicações desenvolvidas em Java sejam independentes da plataforma onde são executadas, ou seja, conferindo característica de portabilidade.

Outra característica dessa linguagem é o fato de, além de ser dinâmica e executável com alto desempenho, possui extensa biblioteca de rotinas que facilita a cooperação com os protocolos baseados em TCP/IP, a citar HTTP e FTP. (CADENHEAD, 2005)

As vantagens que o Java possui englobam sintaxe similar à da linguagem C ou C++, vasto conjunto de bibliotecas (ou APIs), desalocação de memória automática por processo de coleta de lixo, fácil internacionalização por suportar caracteres *Unicode*, especificação simples na linguagem e no ambiente de execução JVM, fácil criação de programas distribuídos e multitarefas, os quais são capazes de executar múltiplas tarefas simultaneamente. (CADENHEAD, 2005)

2.2.2 JAVASCRIPT

Javascript é uma linguagem de programação originalmente implementada como parte dos navegadores Web (*browsers*) na década de 90, com o intuito de que trechos de código pudessem ser executados do lado do cliente e assim, interagir com o usuário sem a necessidade deste *script* passar pelo servidor. Por este motivo, foi criada com o propósito de ser uma linguagem dinâmica, multi-paradigma, interpretada (embora implementações modernas utilizem técnicas de compilação *just-in-time*). (MOZILLA DEVELOPER NETWORK).

Uma JavaScript *engine* - também conhecida como interpretador JavaScript ou uma implementação JavaScript - interpreta código fonte JavaScript e o executa de forma adequada. (*ECMAScript Language Specification*)

Atualmente, o Node.js é uma plataforma para programação JavaScript focada em servidores e processamento sem interface gráfica, por meio do uso de

um interpretador JavaScript desenvolvido pela Google e distribuído no regime de código aberto. (NODEJS)

Segundo Imasters (2011), o objetivo do Node.js é fornecer uma maneira fácil de criar programas de rede escaláveis, visto que não existe um número máximo de conexões simultâneas em um servidor. Esta característica é alcançada ao mudar a forma como a conexão é gerenciada no servidor, consequência esta da arquitetura baseada em eventos do Javascript.

2.3 COMUNICAÇÃO WEB COM AMQP

A troca de informações entre os dispositivos se dá por meio de protocolos de comunicação, que são um conjunto de regras e procedimentos para a transmissão de dados entre equipamentos eletrônicos. (BRITANNICA ACADEMIC, 2014). Dentre os protocolos que possibilitam a comunicação pela rede mundial de computadores, pode-se citar o WebSocket e o AMQP. Esse último está descrito a seguir.

AMQP é sigla para *Advance Message Queuing Protocol*, um protocolo aberto para a transmissão de mensagens entre aplicações. O objetivo deste é padronizar as trocas de mensagens entre componentes de *software*, seguindo os princípios de segurança, confiabilidade e interoperabilidade, ou seja, capacidade do sistema de se comunicar de forma transparente com outro sistema. Ademais, além de o AMQP permitir a comunicação através de redes locais e/ou remotas, característica essa relevante para o objetivo do projeto, o protocolo permite conectar organizações, tecnologias (aplicativos em diferentes plataformas), em tempos distintos (sistemas não precisam estar disponíveis simultaneamente), e diferentes espaços (opera de forma confiável à distância). (RICHARDS, 2011)

Outras características interessantes desse protocolo são a rapidez, a segurança, a confiabilidade, a escalabilidade[1], e a garantia de entrega das mensagens entre aplicações e processos – mesmo quando da falta temporária de comunicação. Esta flexibilidade do AMQP torna o protocolo ideal para monitoramento e controle de sistemas.

Os quatro elementos que protocolo AMQP possui são:

[1] Característica de um sistema que indica sua capacidade de manipular uma porção crescente de trabalho de forma uniforme, ou estar preparado para crescer. (BONDI, 2000)

i. Servidor;
ii. Mensagem;
iii. Produtor;
iv. Consumidor.

O servidor é uma aplicação que aceita conexões de clientes para o roteamento de mensagens, que contém os dados a serem transferidos ou encaminhados. Já o produtor é um aplicativo cria as mensagens a serem enviadas ao consumidor, e este, por sua vez, recebe a mensagem com os dados de interesse.

Além disso, o modelo AMQP define como as mensagens são recebidas, encaminhadas, armazenadas em filas e também como as partes das aplicações são manipuladas. Para que tudo isso ocorra, existem três componentes essenciais: *exchange*, parte do servidor que recebe as mensagens e as encaminhas para as filas, fila de mensagens *(queue)* que é uma entidade que associa as mensagens aos consumidores e *bindings* que são regras de distribuição de mensagens a partir de trocas de filas. (HINTJENS, 2008).

A Figura 2 mostra as filas de mensagens Q1 e Q2, o *exchange*, representado como um x, e os *bindings*, mostrados como as setas entre x e Q1 e Q2.

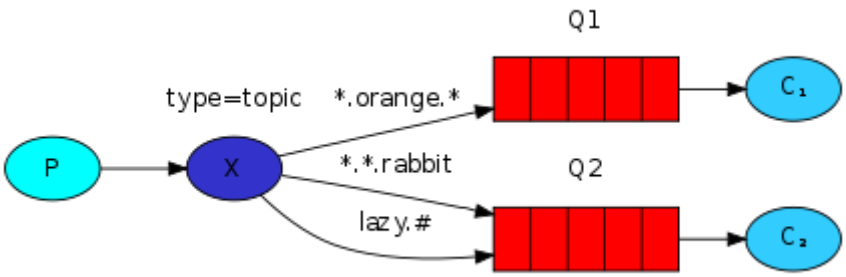

Figura 2 - Elementos do protocolo AMQP
(RABBITMQ, 2014)

Em virtude das características expostas, opta-se pela utilização do protocolo AMQP devido aos atributos de robustez quanto à falta de conectividade de dados, por oferecer mecanismos flexíveis de segurança e qualidade e por garantir a troca de mensagens entre os *softwares.*

2.4 LINGUAGEM DE MODELAGEM UNIFICADA

UML (*Unified Modeling Language*), ou seja, linguagem de modelagem unificada, é uma linguagem visual utilizada para modelar sistemas computacionais por meio do paradigma de orientação a objetos. De acordo com Guedes (2004), nos últimos anos essa linguagem tornou-se padrão de modelagem de *software* adotada internacionalmente pela indústria de Engenharia de Software.

Vale ressaltar que a UML não é uma linguagem de programação, mas sim uma linguagem de modelagem cujo objetivo é auxiliar os engenheiros de *software* a definir as características destes, tais como seus requisitos, sua estrutura lógica, seu comportamento, a dinâmica de seus processos e até mesmo as necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado. Ou seja, UML ajuda os engenheiros na visualização do projeto através de diagramas que mostram a comunicação entre os objetos. Todas essas características são definidas com ajuda desta linguagem antes mesmo de o *software* começar a ser desenvolvido. Portanto, os objetivos da UML são: especificar, documentar e estruturar a lógica para uma melhor visualização e posterior desenvolvimento completo do sistema. (GUEDES, 2004)

Existem inúmeros diagramas na linguagem UML, através dos quais é possível ter múltiplas visões do sistema a ser modelado, bem como maneiras de analisá-lo e modelá-lo. Cada diagrama analisa o sistema sob uma determinada ótica, pois cada um enfoca uma determinada característica. Além disso, a utilização desses diagramas permite que falhas sejam descobertas mais rapidamente, diminuindo assim a possibilidade da ocorrência de erros futuros. Os diagramas escolhidos para esse projeto foram: Diagrama de Classes, Diagrama de Sequências e Diagrama de Atividades

O Diagrama de Classes, representado na Figura 3 é o diagrama mais importante da UML, pois ele define a estrutura das classes, além de estabelecer como as classes se relacionam e como trocam informações entre si.

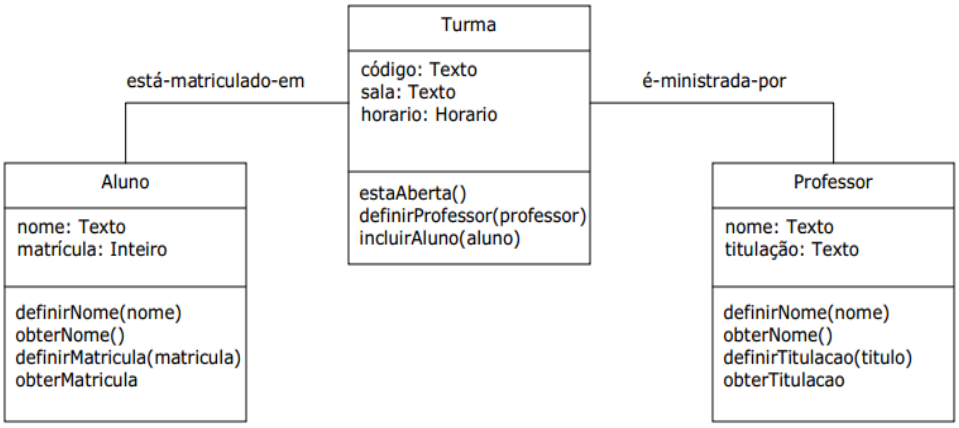

Figura 3 - Exemplo de um Diagrama de Classes
(LES PUC RIO, 2014)

Por sua vez, o Diagrama de Sequências, mostrado na Figura 4, representa a ordem temporal na qual as mensagens são trocadas entre os objetos envolvidos em um determinado processo. Ou seja, identifica o evento gerador do processo modelado, bem como o ator responsável por este evento e então, determina como o processo deve se desenrolar e ser concluído por meio da chamada de métodos disparados por mensagens enviadas entre os objetos. (FOWLER. M, SCOTT.K. 2000)

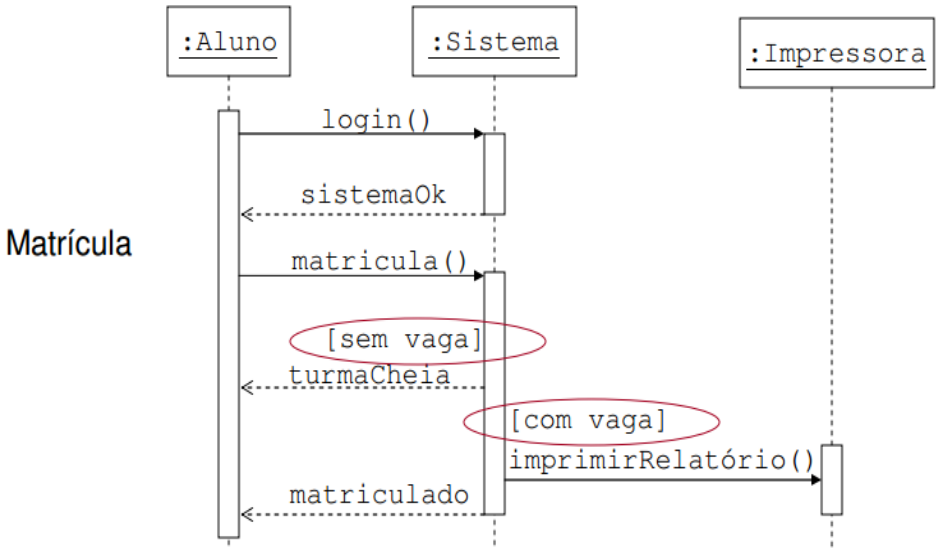

Figura 4 - Exemplo de Diagrama de Sequência
(LES PUC RIO, 2014)

Por fim, o diagrama de atividades é responsável por mostrar o processo do *software* como um fluxo de atividades, ou seja, por meio da sequência dos eventos é

possível visualizar a transição de informação entre os elementos, como se pode ver na Figura 5. (FOWLER. M, SCOTT.K. 2000)

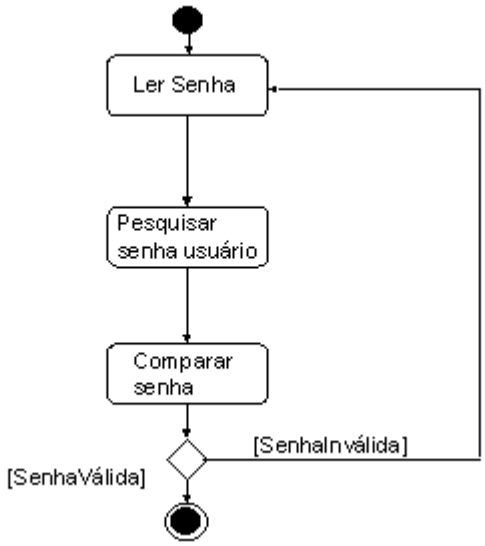

Figura 5 - Exemplo de Diagrama de Atividades
(LEITE, 2000)

3 DESENVOLVIMENTO

Conforme o objetivo proposto no início do trabalho, foi desenvolvido um sistema integrado à Internet capaz de autorizar, monitorar e registrar o acesso de pessoas em ambientes de uso restrito. Foram usados microcomputadores Raspberry Pi, linguagem Java no servidor e Javascript no Raspberry Pi, além do protocolo AMQP. Foi utilizada a linguagem UML como ferramenta de modelagem do sistema.

A opção pelo uso do Raspberry Pi trouxe as seguintes vantagens: baixo custo, facilidade de compra, disponibilização de entradas e saídas digitais de simples acesso e, principalmente, devido ao fácil acesso à Internet que esse *hardware* disponibiliza. Por sua vez, escolheu-se a linguagem Java para o servidor e Javascript no Raspberry Pi, devido às características de simplicidade de programação, flexibilidade, segurança e robustez, as quais são fundamentais para esse trabalho.

O projeto foi desenvolvido para ser implementado em um laboratório e na sala de professores do Departamento Acadêmico de Eletrotécnica do Campus Curitiba da UTFPR. Para tanto, faz-se necessário entender o fluxo de pessoas característico em cada um dos dois ambientes:

a) Primeiramente, em um laboratório há tipicamente a ocorrência de atividades de aula, podendo também acontecer atividades de manutenção do ambiente (incluindo limpeza periódica). Logo, o professor, no seu horário de aula, passará seu crachá – ou digitará sua senha – no módulo de leitor de crachás que se encontrará ao lado da porta do laboratório. Então, a porta do ambiente será liberada e assim permanecerá destravada até o horário programado para o fim da aula, permitindo com que todos os alunos entrem na sala, independente do horário que cheguem (atrasados ou pontuais). Ao final deste período, a porta pode ser travada mediante releitura do crachá do professor, ou automaticamente, de acordo com o horário previamente programado.

   Para a situação em que o professor de determinado horário não compareça e envie um substituto, o mesmo poderá liberar a entrada dos alunos na sala, pois estará em horário de aula, independente do usuário que estará passando o crachá no leitor. O único requisito é que todos os professores que venham a usar o laboratório estejam cadastrados e autorizados no sistema. Fora do

horário de aula, os usuários autorizados poderão destravar a porta mediante o uso do crachá, mas, logo após a sua entrada no ambiente, a porta será travada automaticamente.

b) Na sala dos professores, verifica-se um fluxo mais restritivo de pessoas, similar às metodologias observadas em soluções comerciais. Neste local a porta será liberada apenas por um curto espaço de tempo, concedendo acesso a um único usuário. Não é notada, neste caso, a presença de um estado travado/destravado, como no caso anterior.

3.1 DIAGRAMA DE SEQUÊNCIAS

Nessa seção estão representados os diagramas de sequências que explicam os casos de uso do sistema. Para tal representação foi utilizada a linguagem UML, descrita no capítulo anterior.

3.1.1 AMBIENTE DO TIPO SALA DOS PROFESSORES

O diagrama Figura 6 representa a sequência de ações que são executados no nível do usuário para o caso de um ambiente do tipo sala de professores. Pode-se perceber que existem quatro objetos:

i. Usuário;
ii. Leitor + Raspeberry – neste caso composto pelo módulo *Honeywell* integrado ao Raspberry Pi;
iii. Servidor, responsável por fornecer as informações necessárias para o Raspberry Pi;
iv. Fechadura.

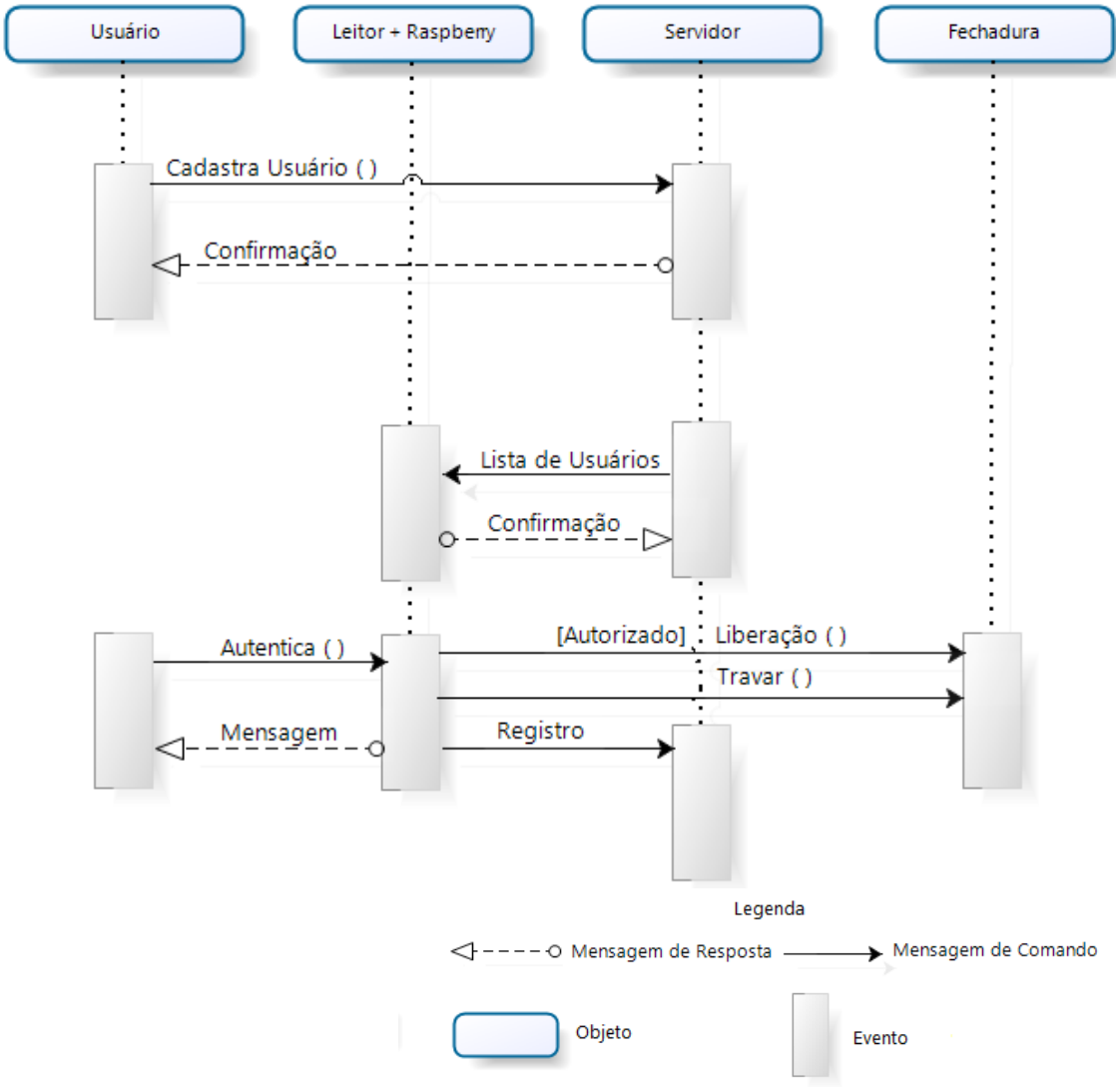

Figura 6 - Diagrama de Sequências da sala de professores
(Imagem do Autor)

Primeiramente, tem-se a fase de cadastramentos, no qual é possível cadastrar, alterar e apagar utilizadores e ambientes no sistema. Após esse cadastro ter sido efetuado, o usuário recebe uma mensagem de confirmação enviada pelo servidor. Além disso, toda vez que uma alteração de cadastro for realizada, o servidor envia uma lista atualizada dos usuários aos componentes afetados pela alteração e, por sua vez, o Raspberry Pi retorna uma mensagem de confirmação de recebimento dessas informações.

A outra fase é a de autenticação, ou seja, o usuário, por meio do código de barras presente em seu crachá ou pela digitação de sua senha, faz uma solicitação no módulo leitor, que por sua vez, mostra uma mensagem se o mesmo está

autorizado a entrar no ambiente em questão. Caso esteja autorizado, o Raspberry Pi envia um comando de liberação à fechadura da porta com o intuito de abri-la. Na sequência, o Raspberry Pi envia um registro para o servidor que armazena o nome do usuário, a data e hora do acesso e se o mesmo foi autorizado ou não a entrar no local desejado. Esse registro, ou troca de mensagens entre o Raspberry Pi e o servidor pode ocorrer devido à utilização do protocolo de comunicação AMQP sempre que houver conexão ativa da rede.

3.1.2 AMBIENTE LABORATÓRIO

O diagrama abaixo representa o caso específico de um laboratório, que possui uma grade de horários, definida com os respectivos professores cadastrados.

A diferença entre esse caso e o anterior é que, quando ocorrer a liberação da abertura da porta, esta não será apenas para o usuário que passou o crachá, pois a porta fica destravada durante todo o horário de aula. Ao acabar a aula, o professor deve passar o crachá novamente no leitor, para que a porta seja então travada. Ou em um caso alternativo, a porta é travada automaticamente após o término do período da permissão.

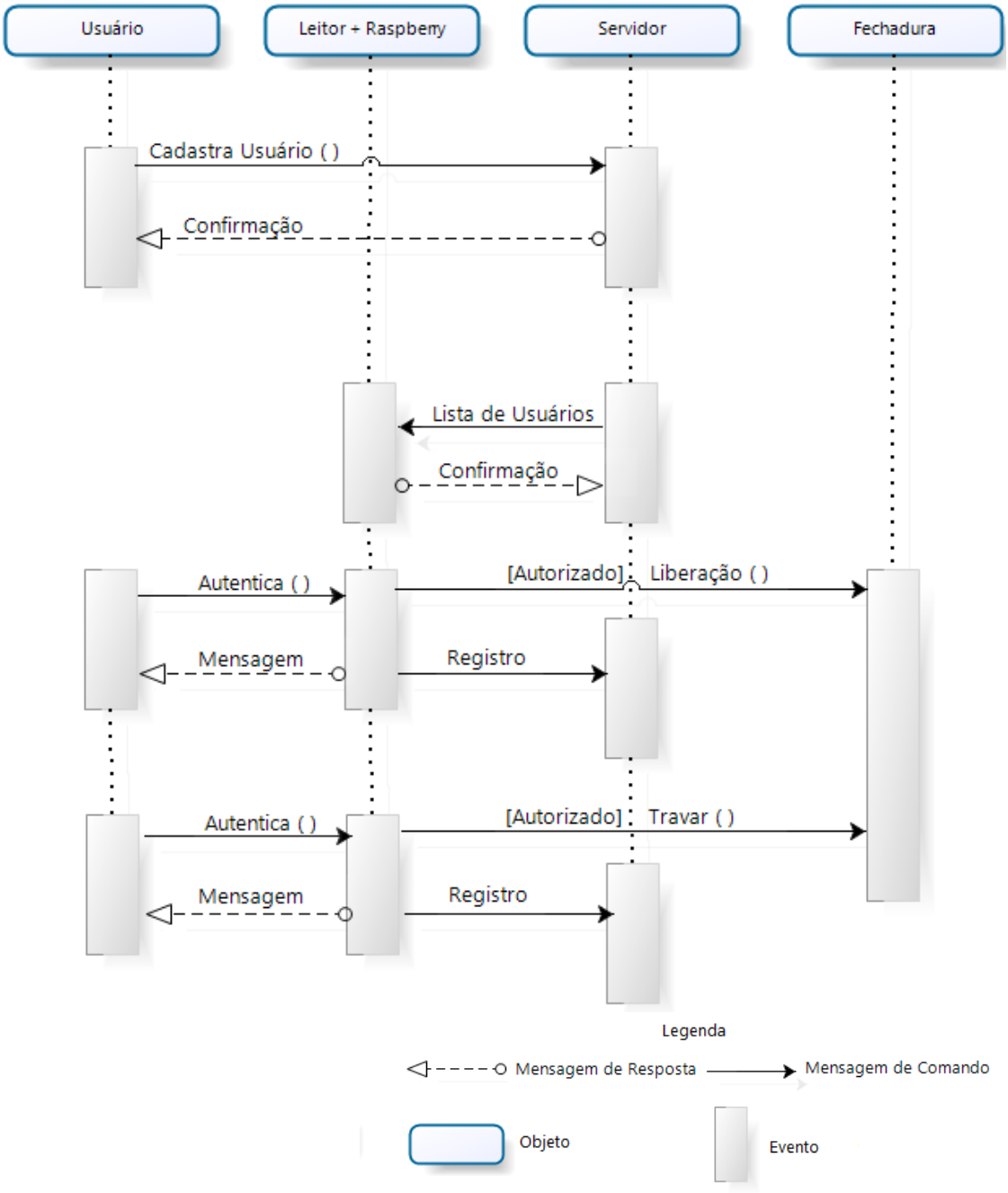

Figura 7 - Diagrama de Sequências do ambiente Laboratório
(Imagem do Autor)

3.2 ARQUITETURA DO SISTEMA

O projeto do sistema de controle de acesso foi baseado em um sistema Web, o qual foi desenvolvido nas linguagens Java e Javascript, e utiliza o protocolo AMQP como forma de comunicação interna. Para entrar no ambiente desejado, o usuário utiliza o código de barras do crachá, e o sistema é responsável pela autenticação e autorização do mesmo. Essa validação é feita localmente por meio do sistema de controle implementado em cada ambiente. Nesse trabalho, a unidade de um ambiente compreende: o Raspberry, o leitor *(Honeywell)*, e a fechadura. No escopo desse trabalho, a palavra rede é usada com o intuito de expressar qualquer rede de comunicação, seja ela local ou remota como, por exemplo, a Internet.

A arquitetura do sistema está descrita na Figura 8.

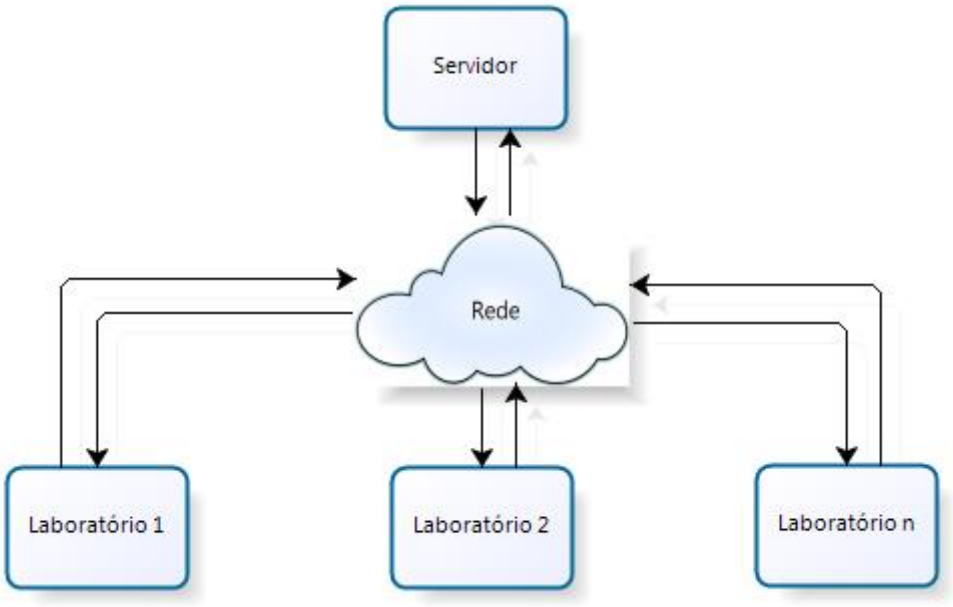

Figura 8 - Arquitetura do Sistema Web
(Imagem do Autor)

O servidor, por meio da rede, troca informações com os ambientes, a fim de transmitir as informações necessárias, de forma a permitir a liberação da porta ao usuário localmente. No servidor, essas informações estão localizadas em um banco de dados global que armazena as autorizações, horários e locais correspondentes a todos os usuários. Em contrapartida, o Raspeberry Pi retém apenas as informações pertinentes ao ambiente onde está instalada, de forma a otimizar o armazenamento de dados.

A Figura 9, mostra em detalhes a composição lógica de um ambiente, o qual é constituído por um Raspberry Pi, uma fechadura e um leitor. O Raspberry Pi faz o papel de controlador local do ambiente, não dependendo diretamente das informações do servidor, motivo pelo qual permite-se que ele trabalhe *offline*.

Para a troca de mensagens com o servidor, o Raspberry Pi usa o protocolo AMQP que, conforme descrito anteriormente, é de suma importância para o trabalho devido às características de confiabilidade, rapidez, segurança e garantia de entrega das mensagens entre aplicações e processos, mesmo que não tenha conexão com a rede. Neste caso, são guardadas todas as informações e transmitidas para o servidor tão logo a conexão com a rede seja reestabelecida.

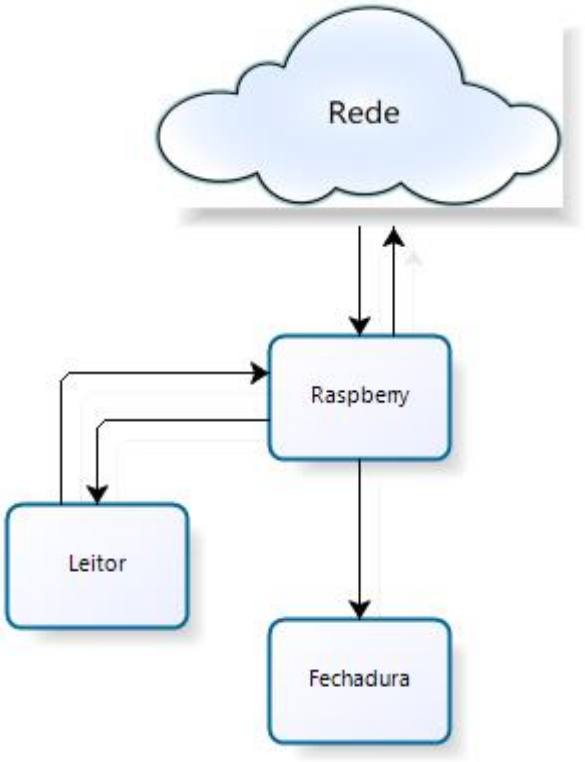

Figura 9 - Comando na porta do ambiente
(Imagem do Autor)

O sistema funciona da forma como aparece na Figura 10. Primeiramente, o usuário deve passar seu crachá no leitor, o qual envia os dados ao Raspberry Pi, que tem a função de autenticar essas informações recebidas. Na sequência, o sistema informa se o usuário em questão está autorizado a entrar no ambiente, com base no horário local e na lista de permissões armazenada. Caso esteja, o sistema verifica se o local pode ser destravado, ou seja, se é possível abri-lo e mantê-lo aberto. Se for, o sistema averigua se o recinto encontra-se destravado; se não

estiver, este é aberto para o usuário e uma mensagem de aviso de abertura da porta é mostrada no leitor. Contudo, caso o ambiente já esteja destravado, o sistema trava o mesmo e exibe uma mensagem de ambiente travado.

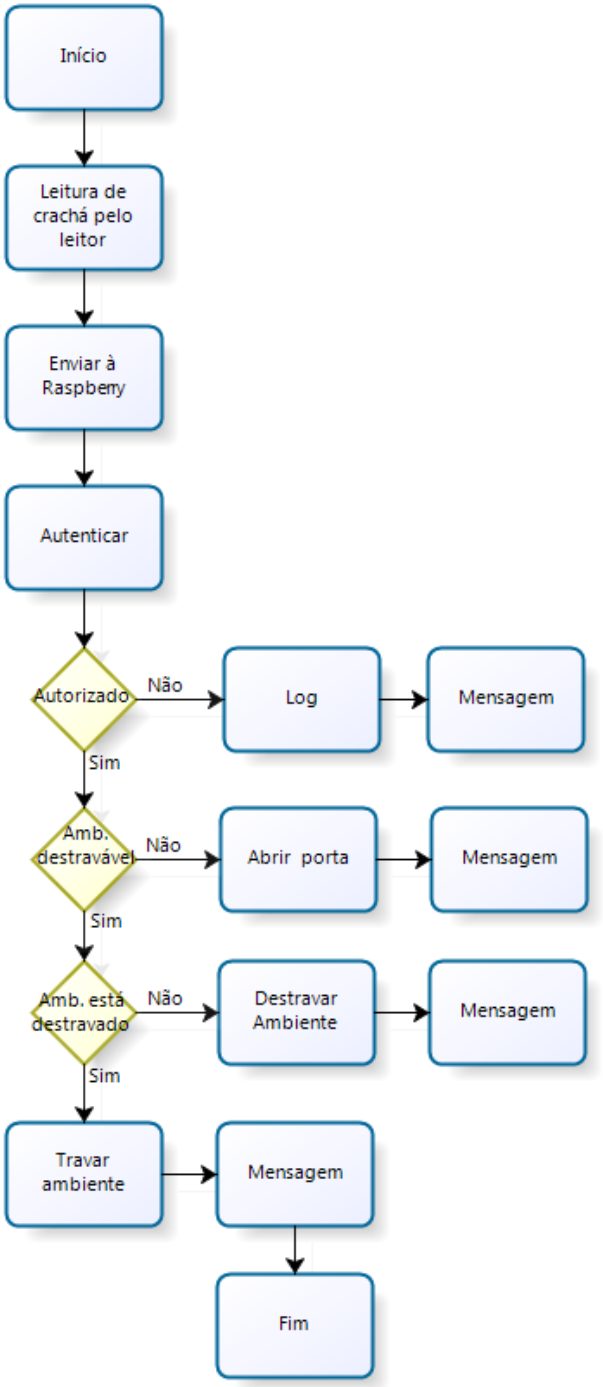

Figura 10 – Diagrama de Atividades do sistema
(Imagem do Autor)

Com relação ao servidor, toda a vez que um item for alterado, um novo comando é enviado aos respectivos ambientes. Por exemplo: ao alterar o código do

laboratório, um novo comando é automaticamente enviado ao Raspberry Pi em questão, atualizando suas referências internas de dados.

3.3 DIAGRAMA DE CLASSES

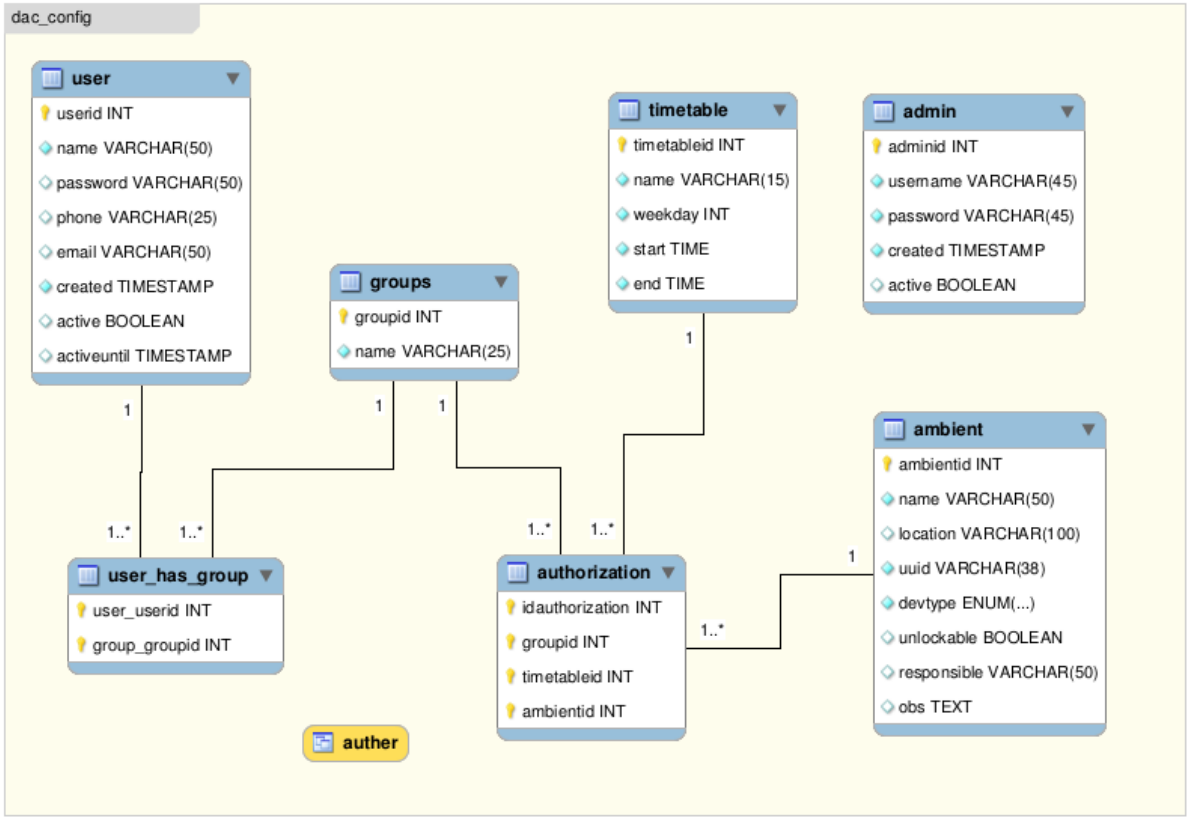

Figura 11 - Diagrama de Classes
(Imagem do Autor)

A figura acima retrata o diagrama de classes do sistema. Pode-se perceber que existem seis principais classes definidas: *User*, *User_has_Group*, *Groups*, *Ambient, Timetable,* e *Authorization*. Além disso, cada uma dessas classes é composta por atributos que são as responsáveis por caracterizá-las. Por exemplo, a classe *Timetable*, que armazena os diferentes horários de autorização possíveis, tem os seguintes atributos: *Timetableid,* representando a identificação única de cada registro; *Name*, representando o nome de cada horário; *Weekday*, representando o dia da semana; e *Start* e *End*; representando os horários inicial e final do horário em questão, respectivamente.

As linhas indicam as relações de dependências que existem entre as classes e, o símbolo 1* mostra que a relação de multiplicidade é de um para muitos, ou seja,

que as classes podem ter um ou mais registros com relações de dependências entre elas.

Por fim, percebe-se que todas as ligações e todas as variáveis convergem para a classe denominada *Authorization*, a qual relaciona essas informações. Isso possibilita determinar se um usuário específico, pertencente a tal grupo, possui autorização de acessar a um determinado ambiente, em um determinado horário.

3.4 HARDWARE

O sistema de *hardware* desenvolvido teve por objetivo prover energia elétrica para os dispositivos, como o módulo *Honeywell*, o Raspberry Pi, a fechadura de eletroímã. Além disso, o sistema possibilita a comunicação dos equipamentos entre si e garante que os comandos enviados pelas aplicações de *software* são atendidos pelos componentes de *hardware*.

Os componentes do sistema podem ser divididos em cinco grupos: alimentação, módulo *Honeywell*, conversor *Buck*, Raspberry Pi, e fechadura. A Figura 12 mostra o diagrama de alimentação do sistema, que será explicado com mais detalhes nas seções abaixo.

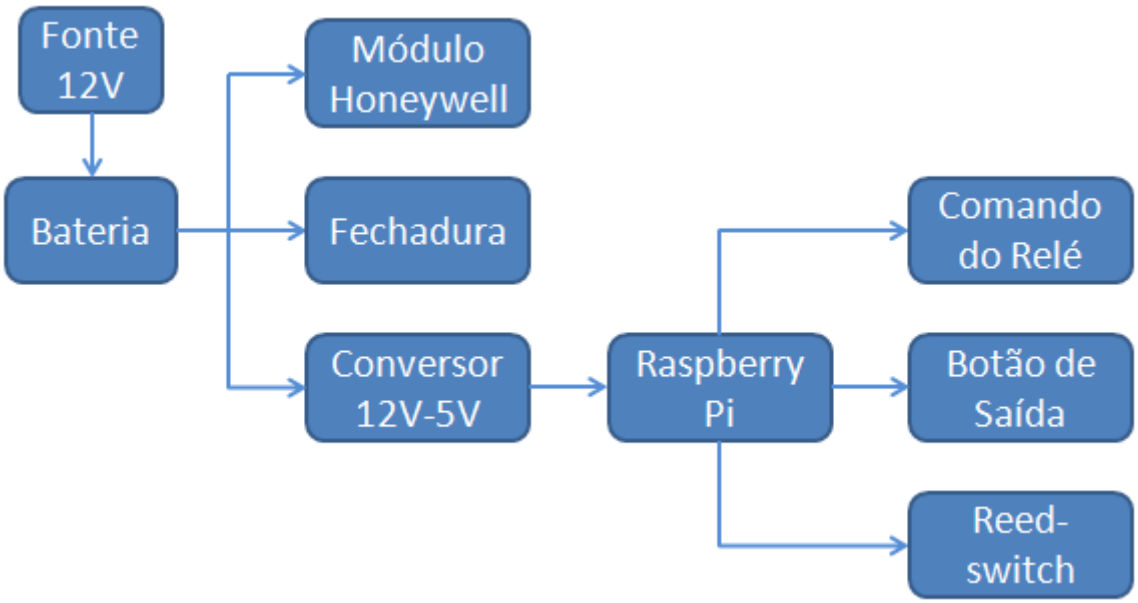

Figura 12 – Diagrama de Alimentação do Sistema
(Imagem do Autor)

3.4.1 ALIMENTAÇÃO

O sistema é alimentado por uma fonte de tensão contínua de 12 V / 5 A ligada à rede elétrica (127 V). A fonte alimenta a placa através de um conector Jack, representado pela notação J1 na Figura 13. Em paralelo à fonte, há uma bateria de 12 V/7 Ah, a qual é responsável por manter o sistema ligado por algumas horas em caso de falta de energia. No polo positivo da fonte está conectado um diodo, de modo que não haja risco de corrente reversa na fonte. A corrente de carga da bateria é limitada por um resistor de 39 Ω. Um arranjo de diodos garante que, ao se alimentar o sistema via fonte ou bateria, não haja perda de energia no resistor de 39 Ω, portanto este só dissipa energia para carga da bateria. O sistema em questão é composto por:

- Uma Fonte de tensão contínua de 12 V / 5 A;
- Três diodos 1N4007;
- Uma bateria de 12 V / 7 Ah;
- Um resistor 39 Ω ½ W.

E, a configuração do mesmo está representada na Figura 13.

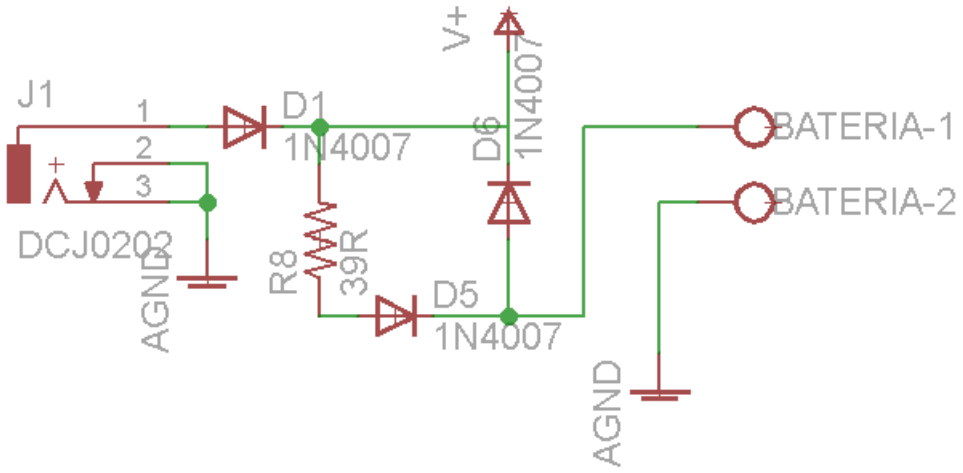

Figura 13 - Alimentação do Sistema
(Imagem do Autor)

3.4.2 MÓDULO HONEYWELL

O módulo *Honeywell* é um sistema embarcado com sistema operacional *Windows CE*, integrado a um leitor de códigos de barra 2D, capaz de identificar, por meio de sua câmera na parte inferior, uma extensa e configurável gama de tipos de

códigos de barras. Possui uma tela LCD *touchscreen* que permite uma interface com o usuário, permitindo tanto a entrada quanto a exibição de dados.

De acordo com os dados operacionais do equipamento, este deve ser alimentado com uma tensão contínua de 12 V, requerendo, no máximo, 860 mA de corrente. Portanto, a alimentação do módulo é realizada por meio do barramento 12 V ao qual é conectada a bateria e, desta forma, possibilita o fornecimento de energia mesmo em caso de falta de energia da rede.

3.4.3 CONVERSOR BUCK

O Raspberry Pi requer uma tensão contínua de alimentação de 5 V, portanto, faz-se necessário um conversor de tensão de 12 V para 5 V. Optou-se por construir um conversor *buck* utilizando o circuito integrado LM2575-05, além dos seguintes componentes:

- Um CI conversor *buck* LM2575-05;
- Um capacitor eletrolítico de 100 µF/25 V;
- Um capacitor eletrolítico de 330 µF/25 V;
- Um indutor SMD de 330 µH;
- Um diodo Schottky 1N5819.

Segundo a folha de dados (TEXAS INSTRUMENTS, 2013) do componente LM2575, este provê todas as funções ativas necessárias para um regulador chaveado *buck*. O conversor pode ser alimentado com uma tensão entre 8 e 40 V e fornece, na saída, uma tensão entre 4,8 e 5,2 V, com corrente de carga de até 1 A. O LM2575 pode operar em uma faixa de temperatura de -40 ºC a 125 ºC.

A ligação utilizada é a recomendada pela folha de dados do componente, conforme indicado na Figura 14.

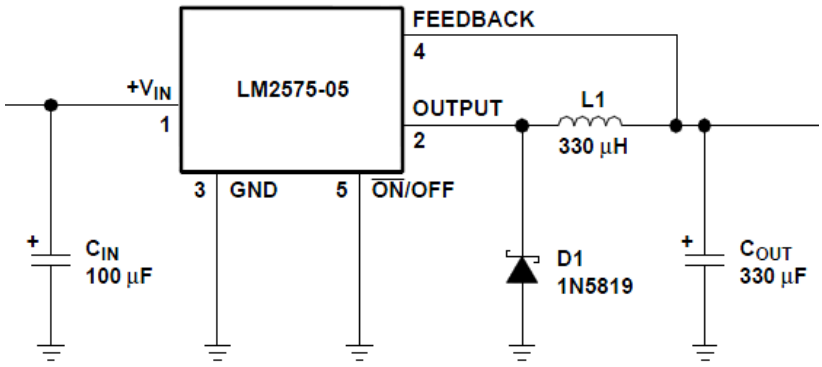

Figura 14 - Esquema de Ligação do Conversor *Buck*
(Fonte: TEXAS INSTRUMENTS, 2013)

O capacitor utilizado na entrada é responsável pela filtragem da tensão elétrica na entrada do conversor, enquanto o capacitor e o indutor na saída do LM2575 são responsáveis pela filtragem da tensão de 5 V. O diodo 1N5819 fornece corrente ao indutor nos períodos em que o LM2575 corta a alimentação, durante o chaveamento. Este componenente suporta uma tensão reversa de 40 V e possui uma tensão direta de 600 mV para 1 A.

O pino 1 do LM2575 recebe a tensão de alimentação de 12 V; o pino 2 fornece a tensão de saída de 5 V; o pino 3 é o ponto de aterramento do componente, sendo a referência para as tensões de entrada e saída; o pino 4 é utilizado no controle da tensão de saída, visto que fornece o *feedback* da tensão de saída para o componente; o pino 5 pode ser utilizado para ligar e desligar o conversor, quando em nível baixo, o componente liga e, quando em nível alto, o componente desliga. O pino 5 não foi utilizado no projeto e portanto aterrado, mantendo o conversor ligado sempre que houver alimentação no seu pino de entrada.

3.4.4 RASPBERRY PI

Conforme explicado na seção 2.1.1, o Raspberry Pi é um microcomputador capaz de rodar aplicações de *software*. Grande parte do controle lógico do projeto está no Raspberry Pi, onde programas e aplicações são responsáveis pela comunicação entre componentes locais e destes com a rede. A lógica de abertura e fechamento da fechadura, autenticação de usuário, histórico de eventos e

transações e comunicação com a rede são exemplos de aplicações programadas no Raspberry Pi.

O Raspberry Pi é alimentado em 5 V por uma entrada micro USB ou por pinos dedicados. O modelo de Raspberry Pi utilizado neste projeto é o modelo B, o qual utiliza uma corrente máxima de 700 mA, dependendo do funcionamento. As entradas e saídas digitais, ou seja, os pinos de GPIO, suportam 50 mA de corrente distribuída entre os pinos, enquanto um pino individual suporta, no máximo, uma corrente de 16 mA.

O módulo do Raspeberry Pi utiliza adicionalmente os seguintes componentes:

- Um Raspberry Pi;
- Um resistor de 1 kΩ;
- Dois resistores de 10 kΩ;
- Uma botoeira;
- Um *reed-switch*.

Dois pinos de GPIO são utilizados como entradas no Raspberry Pi, um pino (CHAVE) para o botão que fica dentro do ambiente controlado, cuja função é destrancar a porta pelo lado de dentro, e outro pino (RS) para o *reed-switch*, o qual possui a função de verificar quando a fechadura for destravada. Além disso, outro pino (GPIO25) é utilizado para saída, o qual envia o comando de destravamento da porta. O Raspberry Pi possui um pino que fornece uma tensão de 3,3 V para que, em conjunto com os pinos de entrada, seja possível a leitura de acionamento de chaves. Os pinos de entrada e saída operam com uma tensão de 3,3 V.

O *reed-switch* é um dispositivo dividido em duas partes, uma delas consiste em um pequeno ímã, a qual é acoplada à porta. A outra parte possui uma chave que é acionada na presença de um campo magnético. Nesse projeto, os terminais da chave são conectados a um optoacoplador, o qual é conectado ao Raspberry Pi. Desta maneira, o Raspberry Pi detecta cada mudança de estado do *reed-switch*.

A Figura 15 mostra o conjunto de pinos que conecta o Raspberry Pi aos circuitos descritos. Os pinos GPIO24 e GPIO27 permitem sinalização via LEDs conforme configuração realizada no programa do Raspberry Pi. Na placa acoplada ao Raspberry Pi também foram previstas conexões a esses pinos.

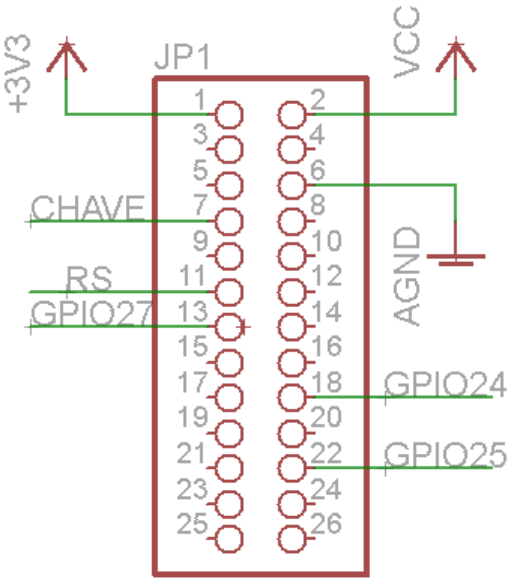

Figura 15 - Esquema de Ligação do Raspberry Pi
(Imagem do Autor)

A Figura 16 mostra o esquema de acionamento dos pinos CHAVE e RS do Raspberry Pi conforme acionamento do botão de saída e do *reed-switch*, respectivamente.

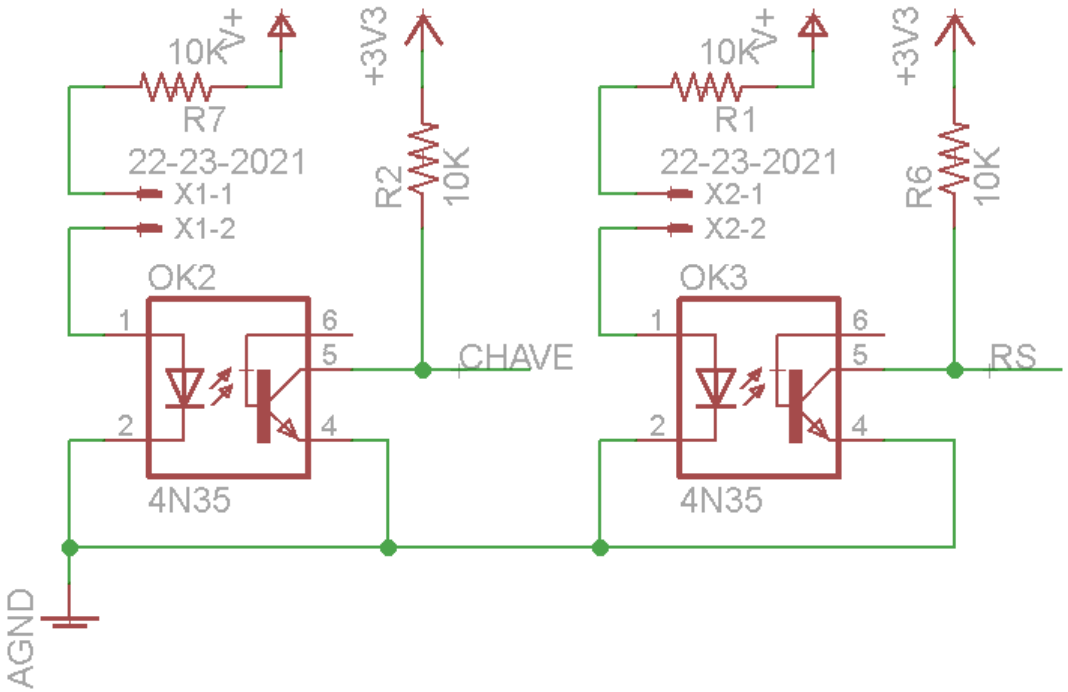

Figura 16 - Leitura dos Pinos CHAVE e RS do Raspberry Pi
(Imagem do Autor)

3.4.5 FECHADURA

A Figura 17 mostra o esquema de ligação da fechadura de eletroímã. A saída do Raspberry Pi (GPIO25) envia um sinal elétrico para destravamento da porta, o qual entra pelo pino 1 do optoacoplador 4N35. A corrente acende o LED dentro do 4N35, o qual libera a passagem de corrente entre os pinos 5 e 4. A corrente que passa pelos pinos 5 e 4 do optoacoplador é a corrente de base do transistor BC548. Com o transistor BC548 saturado, há a passagem de corrente pela bobina do relé AY1RC2, fazendo-o comutar. A chave NF (Normalmente Fechado) do relé abre e interrompe a passagem de corrente pelo eletroímã da fechadura, desmagnetizando-o. Sem força magnética, a fechadura é destravada, permitindo a abertura da porta.

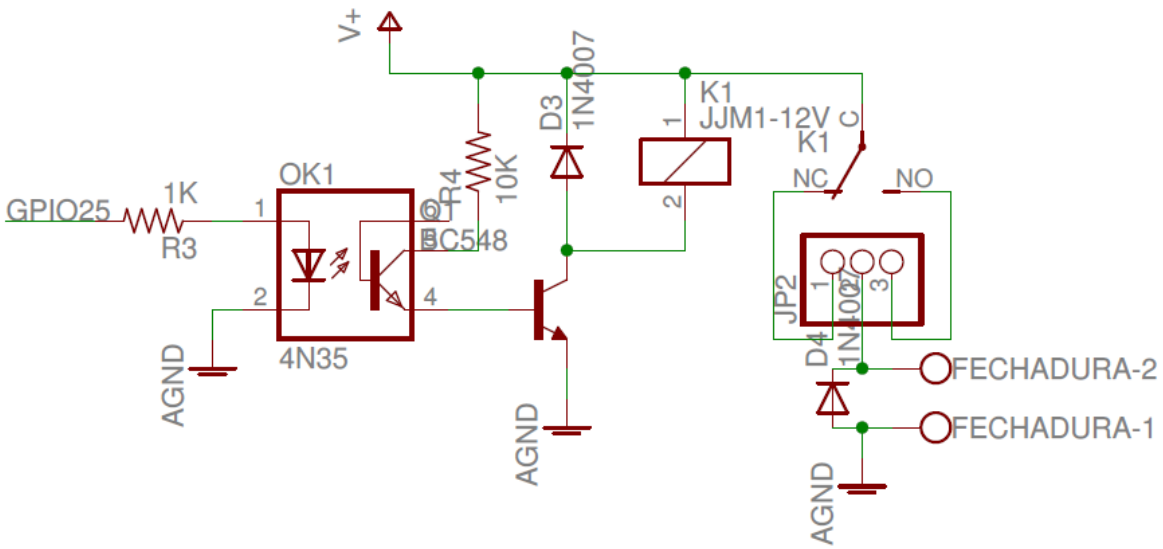

Figura 17 - Esquema de Ligação da Fechadura de Eletroímã
(Imagem do Autor)

O esquema de ligação descrito acima é composto pelos seguintes elementos:

- Uma fechadura de eletroímã 12 Vdc / 345 mA;
- Um relé AY1RC2 de 12 V;
- Dois diodos de roda livre 1N4007;
- Um transistor BC548;
- Um optoacoplador 4N35;
- Um resistor de 10 kΩ.

3.5 RECURSOS

Os recursos necessários para a realização desse projeto foram:

1. Recursos humanos: integrantes do projeto, gestão do projeto, modelagem e desenvolvimento do *software* e desenvolvimento do *hardware*;
2. Recursos materiais: dois laboratórios da UTFPR (D-105 e sala de professores), fechadura, Raspberry Pi, fonte de tensão, componentes eletrônicos, baterias, leitor de cartão (módulos *Honeywell* doados pela UTFPR), caixas e computador;
3. Recursos de software: sistemas operacionais Linux Ubuntu e Linux Raspbian, Servidor J2EE Glassfish, Máquina virtual Java, Interpretador Javascript Node.JS, Banco de dados MySQL, Servidor AMQP RabbitMQ e diversas bibliotecas relacionadas às linguagens de programação citadas;
4. Recursos financeiros: O projeto foi selecionado pelo Edital de Apoio à Execução de Trabalho de Conclusão de Curso e recebeu um financiamento total de 1000 reais proveniente da empresa OMS Engenharia. Os custos iniciais para execução do projeto constam na tabela a seguir:

TIPO	VALOR UNITÁRIO	QUANT.	VALOR TOTAL
Fechadura	R$ 160,00	2	R$ 320,00
Raspberry Pi	R$ 115,96	2	R$ 231,92
Caixas	R$ 35,00	2	R$ 70,00
Fonte de tensão	R$ 29,99	4	R$ 119,96
Componentes Eletrônicos	--------	Diversos	R$ 251,68
Baterias	R$ 63,47	2	R$ 126,94
Impressão do trabalho	R$ 11,90	9	R$ 107,10
Prototipagem das placas	R$ 47,80	5	R$ 239,00
Total			R$ 1.466,60

Tabela 1 - Custos do projeto
(Tabela do Autor)

4 MEMORIAL DESCRITIVO E TESTES

Nesse capítulo apresenta-se a implementação do sistema de controle de acesso no laboratório D-105 e na sala dos professores, ambos situados no Campus Curitiba da UTFPR, e os testes realizados nesse sistema.

4.1 MEMORIAL DESCRITIVO

Nessa seção são explanados os resultados da implementação tanto da parte de *hardware* quanto de *software* do projeto.

4.1.1 MEMORIAL DESCRITIVO DE HARDWARE

A placa de circuito impresso, mostrada na Figura 18, foi projetada utilizando-se o programa EAGLE 6.6.0. A aplicação de *software* EAGLE permite a criação tanto do esquemático quanto do projeto físico da placa. A Figura 19 mostra o esquemático da placa.

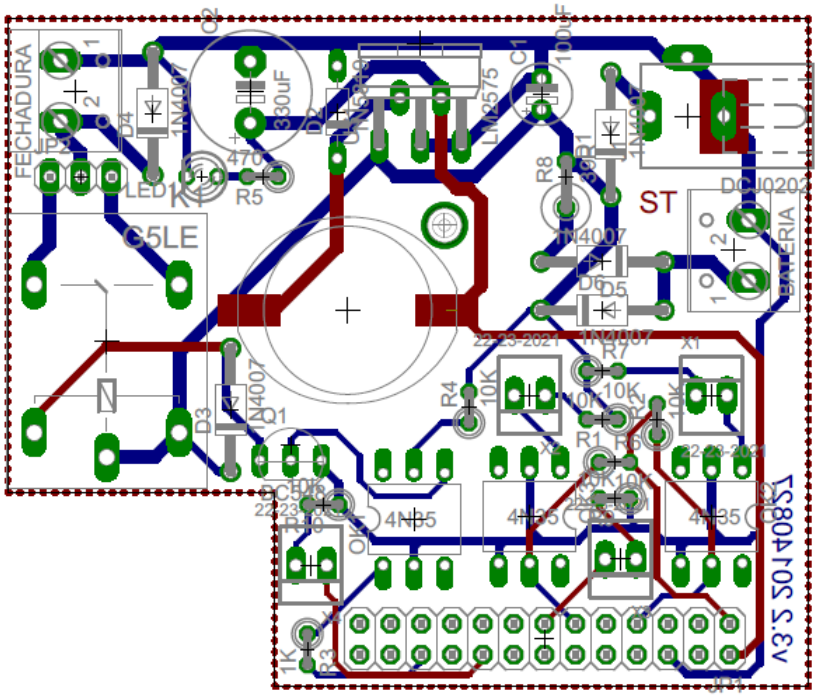

Figura 18 - Projeto da Placa de Circuito Impresso
(Imagem do Autor)

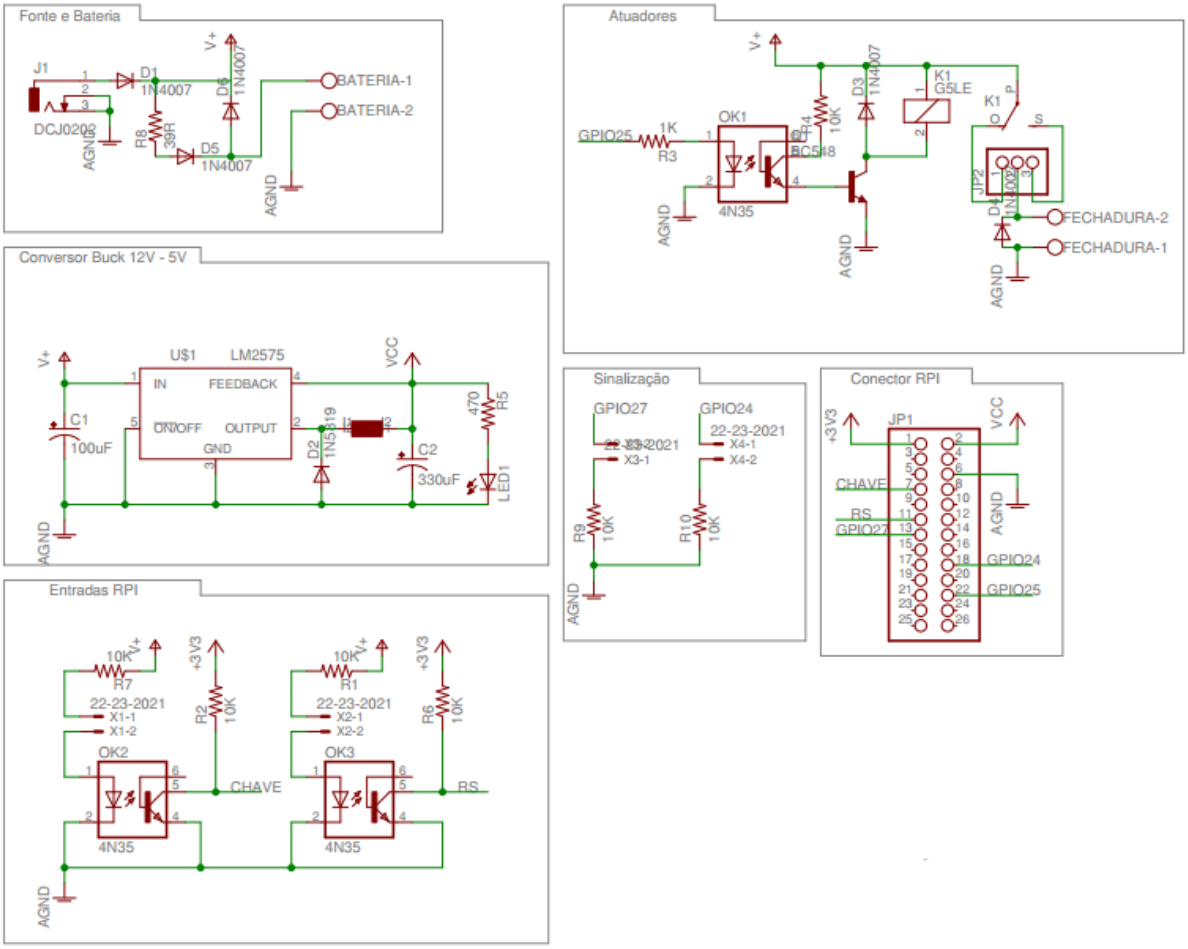

Figura 19 - Esquemático da Placa de Circuito Impresso
(Imagem do Autor)

O tamanho e formato da placa foram projetados para encaixar no Raspberry Pi, conforme mostra a Figura 20 e a Figura 21.

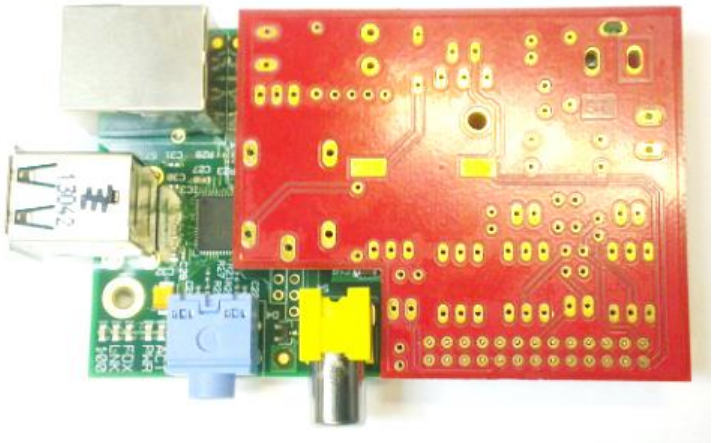

Figura 20 - Placa montada no Raspberry Pi
(Imagem do Autor)

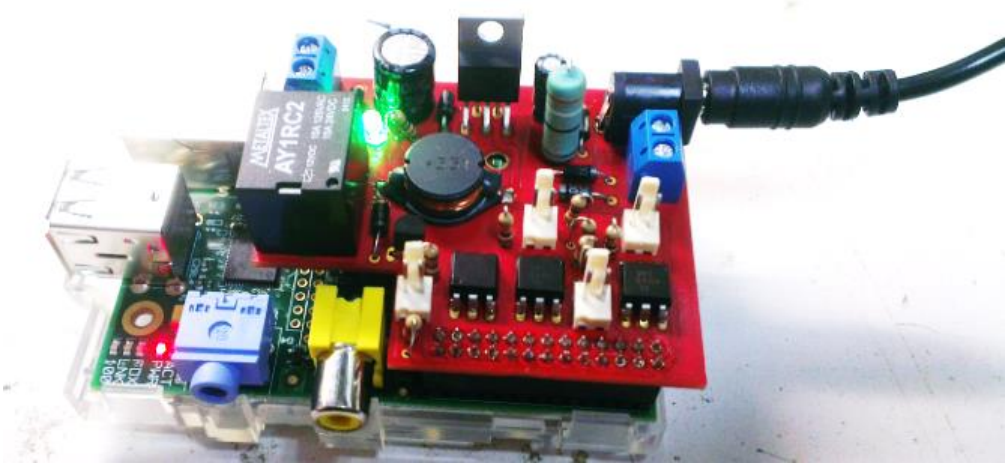

Figura 21 - Energização da Placa e Raspberry Pi
(Imagem do Autor)

Para acomodar o Raspberry Pi, a placa, a fonte, a bateria e os cabos, utilizou-se uma caixa de plástico, mostrada na Figura 22. A Figura 23 mostra a caixa já fixada na parede da sala dos professores no DAELT.

Figura 22 - Caixa com os Componentes
(Imagem do Autor)

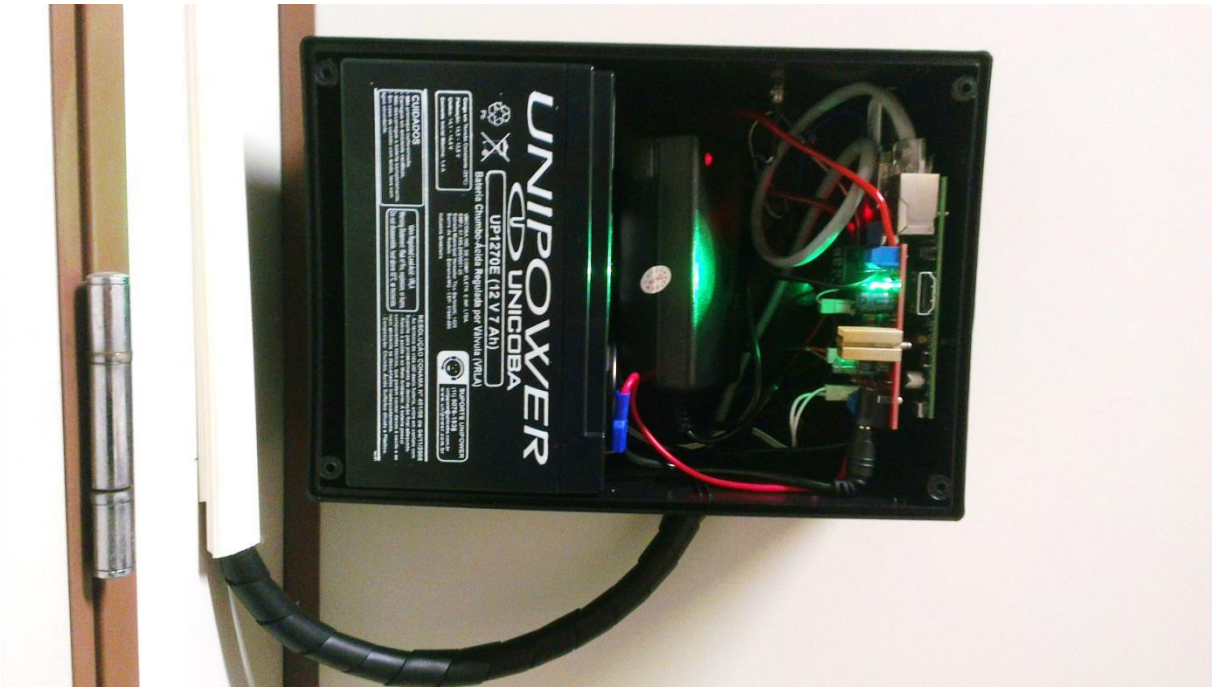

Figura 23 - Caixa fixada no DAELT
(Imagem do Autor)

Os módulos para controle de acesso foram montados em dois locais: na sala dos professores do DAELT e no laboratório D-105. No DAELT, o módulo foi instalado na porta da sala dos professores. Um trabalho similar a este já havia sido iniciado na sala dos professores, portanto aproveitou-se canaletas, passagem de cabos e fechadura. Entretanto, a fechadura instalada anteriormente era diferente daquela considerada no sistema projetado. Análises da fechadura instalada foram realizadas e concluiu-se que uma rotina diferente no *software* permitiria a utilização desta, a qual se trata de uma fechadura eletromecânica. Uma particularidade dessa instalação é que a caixa foi fixada na sala ao lado por facilidade de acesso à energia elétrica. Por este motivo, o botão de saída foi fixado fora da caixa e perto da porta.

No laboratório D-105, a fechadura utilizada foi a eletromagnética modelo FS150 do fabricante Automatiza. A caixa de plástico foi colocada ao lado da porta, permitindo que o botão de saída estivesse localizado na própria caixa. Um dos módulos *Honeywell,* que a UTFPR cedeu para a realização do trabalho, estava queimado e em curto circuito, o que acabou queimando alguns componentes da placa, os quais foram substituídos, juntamente com o módulo.

4.1.2 MEMORIAL DESCRITIVO DE SOFTWARE

Conforme a arquitetura planejada para o sistema de controle de acesso, foram utilizados o banco de dados MySQL e a linguagem Java para o desenvolvimento da aplicação do servidor. O banco de dados tem a função de persistência dos dados e configurações, as quais são acessadas e utilizadas pelo *software* desenvolvido em Java. Além disso, usou-se o programa RabbitMQ[2], também instalado no servidor, com o intuito de prover o serviço de troca de mensagens do protocolo AMQP. Por fim, a aplicação do Raspberry Pi foi desenvolvida em Javascript, a qual comunica-se com o servidor através deste protocolo.

A Figura 24 exemplifica a forma de implementação do projeto.

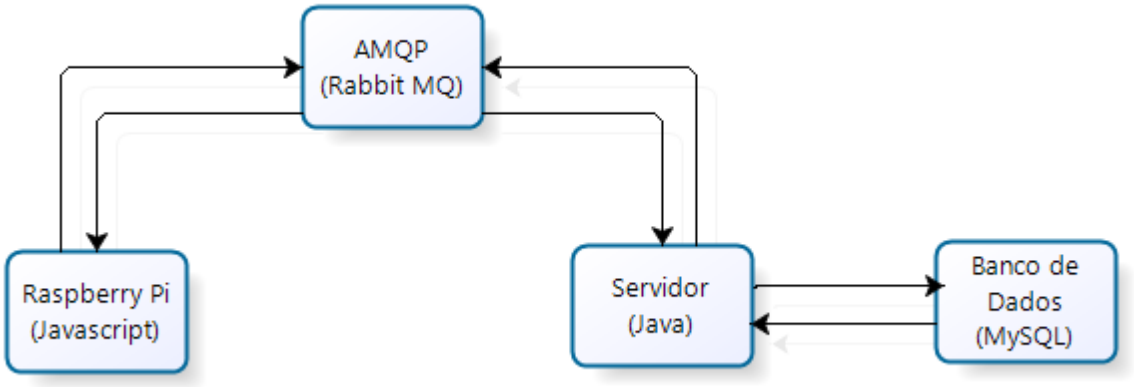

Figura 24 - Implementação do AMQP com o RabbitMQ
(Imagem do Autor)

Com o objetivo de tornar o sistema modular e flexível, o *software* do Raspberry Pi foi concebido de forma modular, onde cada módulo realiza uma tarefa específica. Para o caso do sistema de acesso, os seguintes módulos foram desenvolvidos e orquestrados de maneira a obter as funcionalidades desejadas:

- Módulo de autenticação: responsável por autenticar os usuários, com base em configurações internas com permissões, horários e senhas;
- Módulo de entradas e saídas: responsável por comandar e gerenciar o estado das entradas e saídas de *hardware* do Raspberry Pi;

[2] RabbitMQ é um *message broker* – um intermediário para troca de mensagens. Oferece às apicações uma plataforma comum para enviar e receber mensagens, e às mensagens um lugar seguro para serem armazenadas até a sua entrega. (RABBITMQ, 2015)

- Módulo servidor Web: responsável por gerar páginas Web simples em HTML, que são requisitadas e exibidas pelo leitor *Honeywell*.
- Módulo de troca de mensagens: responsável por gerenciar a troca de mensagens com o servidor, utilizando o protocolo AMQP.
- Módulo de controle: responsável por armazenar o estado do sistema, bem como processar a lógica do mesmo, coordenando o funcionamento dos outros módulos.

A interface com o usuário dá-se apenas por meio do leitor *Honeywell*, o qual exibe páginas HTML geradas de acordo com o estado corrente do sistema, mostrando o nome do ambiente, a concessão ou não do acesso, o nome do usuário, a hora local, entre outras informações. A Figura 25 mostra a tela inicial de um ambiente de teste.

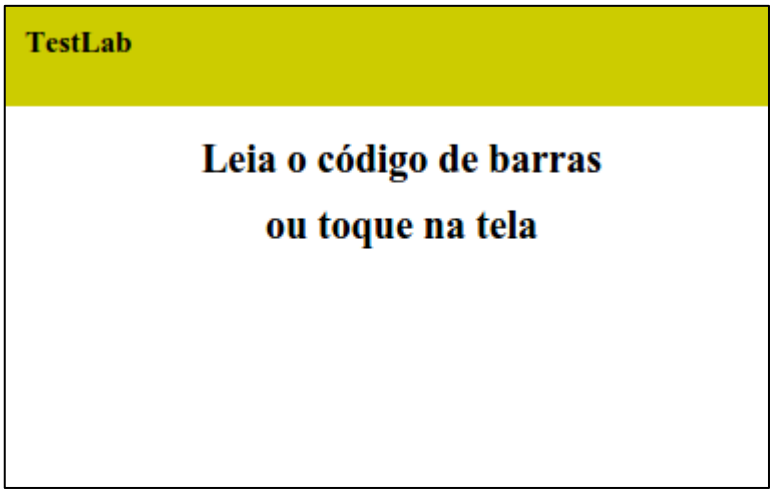

Figura 25 - Tela Inicial do Leitor *Honeywell*
(Imagem do Autor)

Já o papel do servidor é o de permitir o cadastro, modificação e exclusão de usuários e ambientes, a configuração de permissões, e a visualização de eventos de forma simplificada e centralizada, gerenciando assim, o sistema como um todo.

Para tanto, a interface com o usuário administrador dá-se por meio de um navegador de internet, o qual, após autenticação deste usuário, acessa e exibe páginas Web contendo as informações relevantes, conforme demonstrado na Figura 26. A Figura 26 mostra a tela de usuários cadastrados. A Figura 27 e a Figura 28 representam as páginas de cadastro de novos usuários e ambientes, respectivamente.

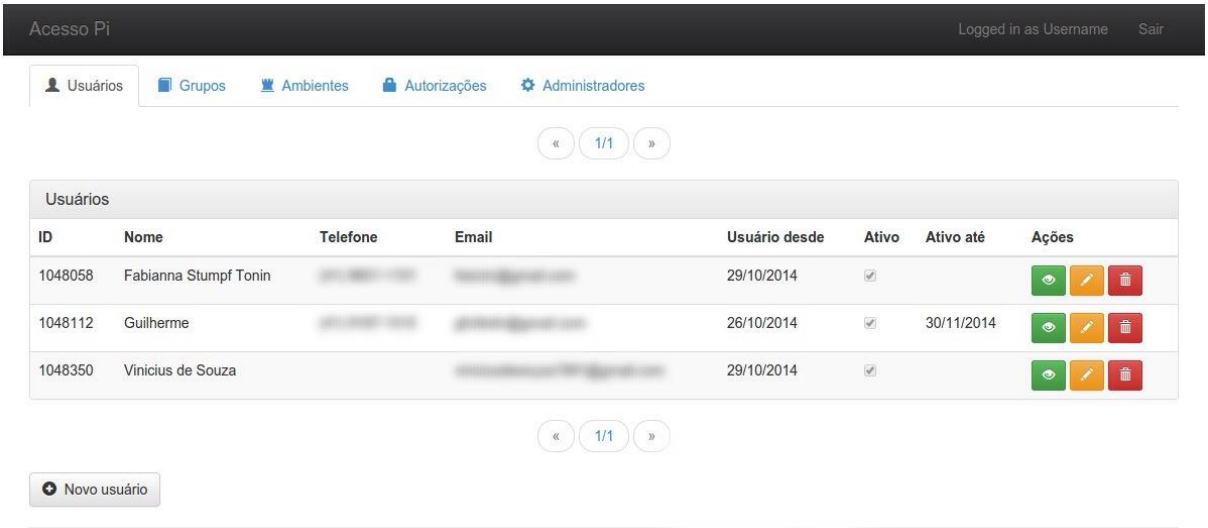

Figura 26 - Tela de usuários cadastrados
(Imagem do Autor)

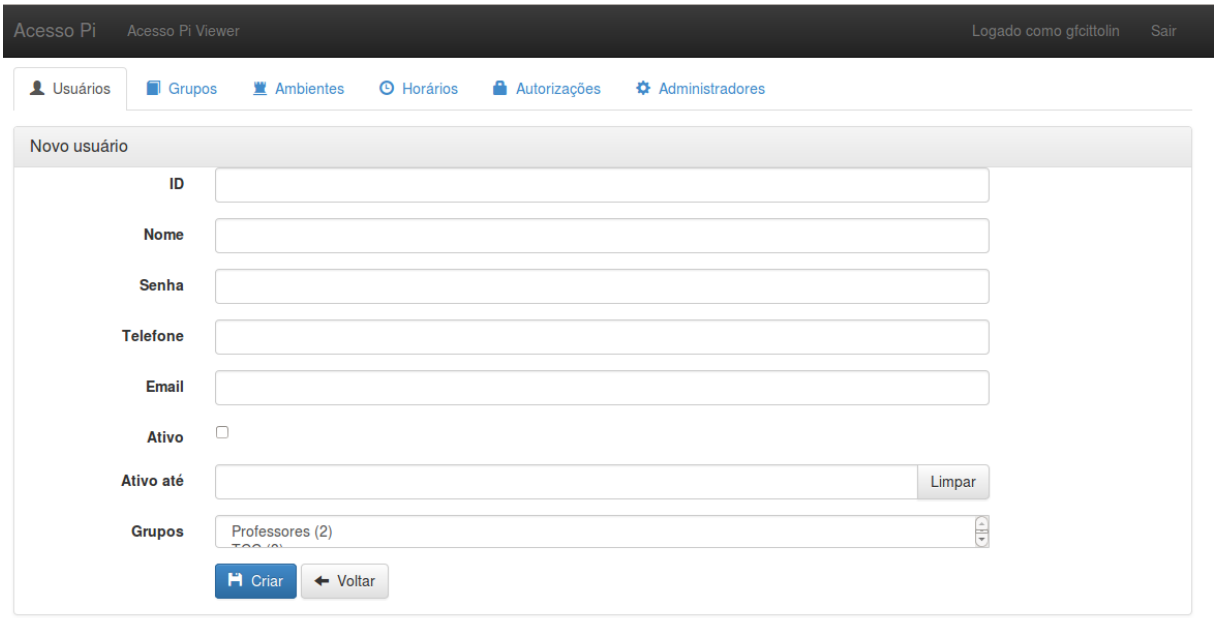

Figura 27 - Tela de Cadastro de Novos Usuários
(Imagem do Autor)

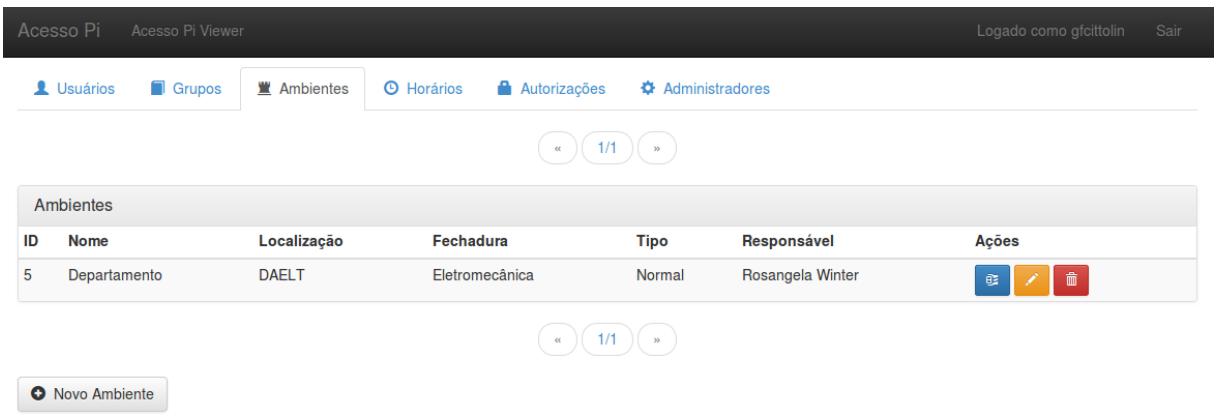

Figura 28 - Tela de Cadastro de Ambientes
(Imagem do Autor)

A tela de autorizações é mostrada na Figura 29. A Figura 30 mostra o registro de eventos do sistema.

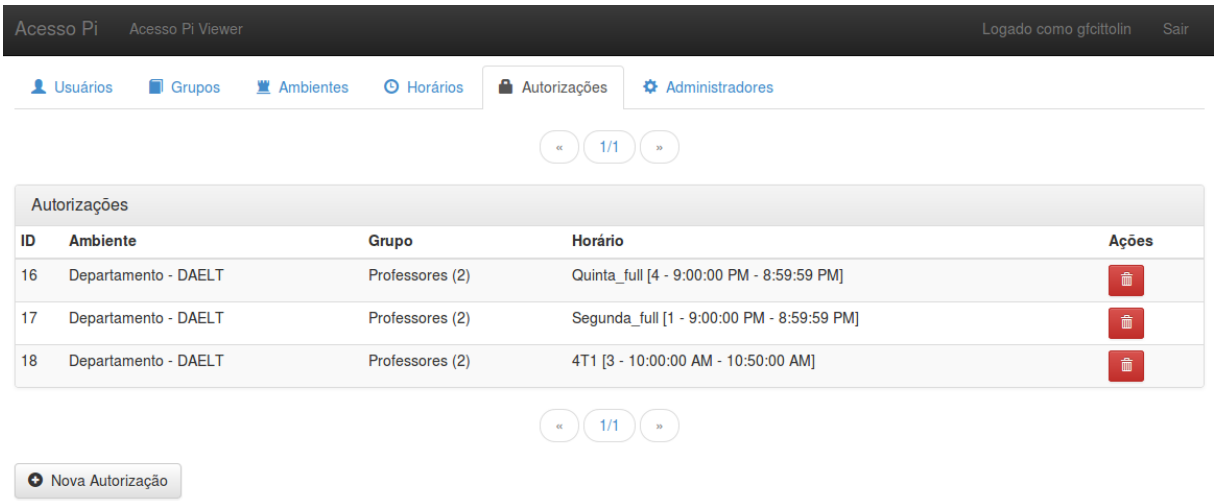

Figura 29 - Tela de Autorizações
(Imagem do Autor)

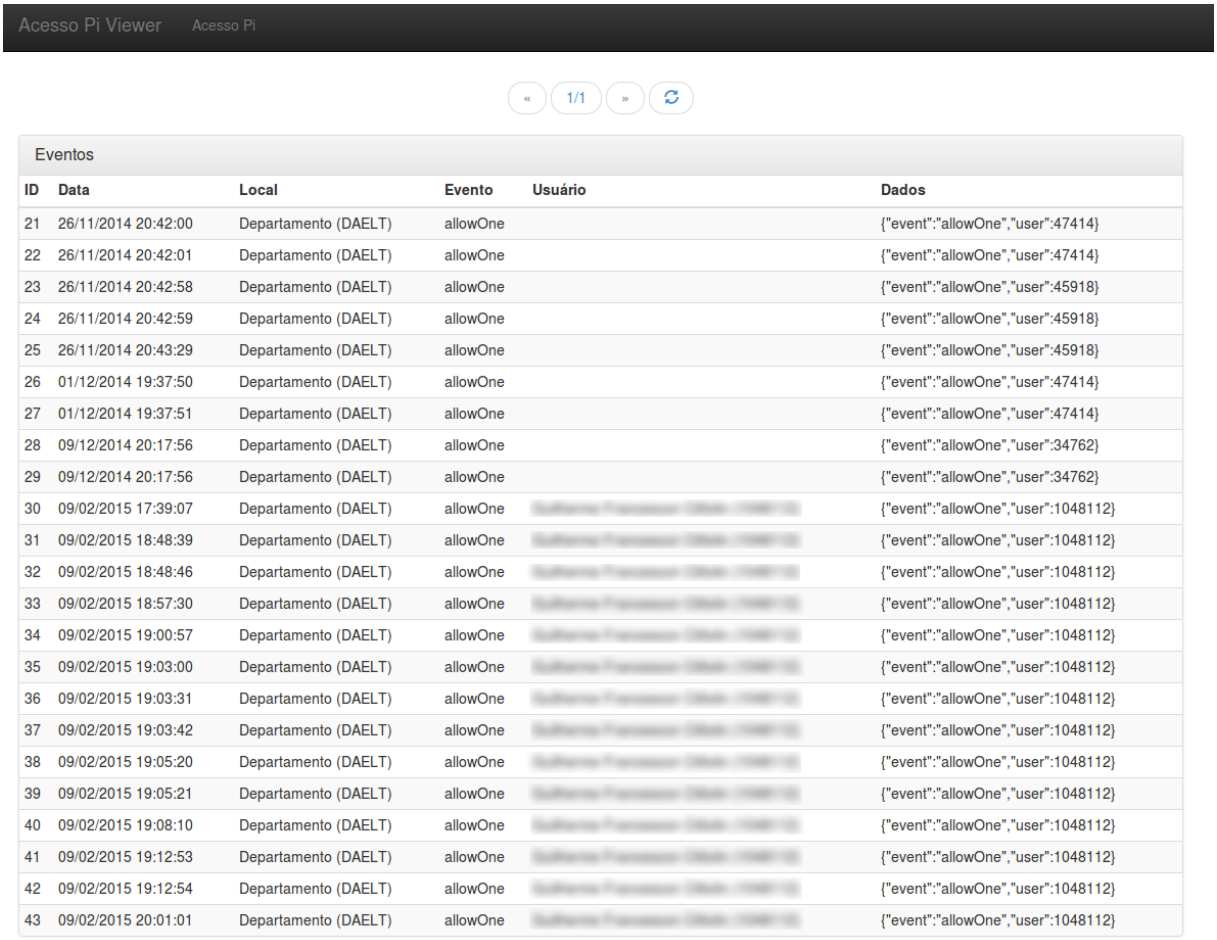

Figura 30 - Tela de Registro de Eventos
(Imagem do Autor)

4.2 TESTES

Nessa seção são abordados os três principais casos de uso que foram testados com o intuito de verificar e validar as funcionalidades propostas neste trabalho.

4.2.1 CASO 1 – USUÁRIO NÃO CADASTRADO E UTILIZAÇÃO DO CRACHÁ

Nesse caso tem-se um usuário que não está cadastrado no sistema, mas que tenta com o seu crachá obter acesso ao ambiente.

O caso descrito foi testado nas combinações rede (conectada e desconectada), energia (operando à bateria e conectada à energia) e ambiente (sala de aula e sala dos professores), conforme mostra a tabela a seguir.

Rede	Energia	Ambiente
Conectado	Operando a bateria	Sala de aula
Conectado	Conectado à energia	Sala de aula
Conectado	Operando a bateria	Sala dos professores
Conectado	Conectado à energia	Sala dos professores
Desconectado	Conectado à energia	Sala dos professores
Desconectado	Operando a bateria	Sala dos professores
Desconectado	Conectado à energia	Sala de aula
Desconectado	Operando a bateria	Sala de aula

Tabela 2 - Combinações para testes
(Tabela do Autor)

Em todas as combinações testadas, obteve-se o resultado esperado, que era o de não permitir o acesso do usuário ao ambiente. Além disso, nos casos em que não havia conexão à rede de dados, o registro dos eventos só ocorreu no servidor após o reestabelecimento da conexão, conforme previsto.

4.2.2 CASO 2 – USUÁRIO CADASTRADO E ACESSO FORA DO SEU HORÁRIO

Nesse caso tem-se um usuário cadastrado no sistema, mas que tenta com o seu crachá obter acesso ao ambiente, fora do seu horário de permissão.

O caso descrito foi testado nas combinações rede (conectada e desconectada), energia (operando à bateria e conectada à energia) e ambiente (sala de aula e sala dos professores), conforme mostra a Tabela 2.

Em todas as combinações testadas, obteve-se o resultado esperado que para a sala dos professores era a não permissão do acesso à sala, pois o usuário estava fora do horário previsto. E, no caso da sala de aula, o usuário conseguiu entrar na sala, porém não conseguiu desbloquear a mesma – que consiste em deixar a sala destravada até o fim do horário da aula em questão.

Além disso, nos casos em que não havia conexão à rede de dados, o registro dos eventos só ocorreu no servidor após o reestabelecimento da conexão, conforme previsto.

4.2.3 CASO 3 – USUÁRIO CADASTRADO E DENTRO DO SEU HORÁRIO

Nesse caso tem-se um usuário cadastrado no sistema, o qual com tenta com o seu crachá obter acesso ao ambiente, dentro do seu horário de permissão.

O caso descrito foi testado nas combinações rede (conectada e desconectada), energia (operando à bateria e conectada à energia) e ambiente (sala de aula e sala dos professores), conforme mostra a Tabela 2.

Em todas as combinações testadas, obteve-se o resultado esperado que para a sala dos professores era a ter o acesso concedido à sala, pois o usuário além de ser cadastrado, estava no horário previsto. E, no caso da sala de aula, o usuário teve acesso ao ambiente, pois a porta foi desbloqueada, ou seja, ela fica aberta durante todo o horário da aula. Ao fim desse horário, a porta é automaticamente travada.

Além disso, nos casos em que não havia conexão à rede de dados, o registro dos eventos só ocorreu no servidor após o reestabelecimento da conexão, conforme previsto.

5 CONCLUSÕES

Este trabalho propôs o desenvolvimento de um sistema de acesso integrado à Internet, permitindo a autorização, o monitoramento e o registro da entrada e saída de pessoas em ambientes de uso restrito.

Para atender aos requisitos do projeto, foi realizada uma pesquisa bibliográfica com a finalidade de buscar recursos e tecnologias já existentes, relacionados tanto ao *software* quanto ao *hardware*. Encontrou-se como potencial plataforma de *hardware* o Raspberry Pi, bem como a linguagem Javascript e o protocolo AMQP como componentes de *software* que atendessem aos pré-requisitos iniciais. A possibilidade de transferência de dados não simultânea desse protocolo foi de suma importância para que o requisito de robustez do sistema à indisponibilidade conectiva fosse atendido.

Buscou-se então especificar a arquitetura do sistema, baseado nas escolhas de *software* e *hardware*. Foi especificada uma placa de interfaces para o Raspberry Pi, responsável pelo acoplamento elétrico entre os sinais externos e internos, bem como por fornecer alimentação ininterrupta por meio do uso de uma bateria. Também foram definidos os dados necessários e como eles se relacionariam, de forma a possibilitar a autenticação e autorização de usuários, de acordo com as necessidades de um sistema de acesso. Além disso, foi desenvolvida a metodologia de comunicação entre os dispositivos, de forma que o sistema fosse robusto à indisponibilidade de comunicação, além de torná-lo modular.

Entre as dificuldades encontradas, pode-se citar principalmente a disponibilidade do laboratório D-105 para a implantação do projeto, o desafio de especificar as partes necessárias do sistema, tanto em *software* quanto em *hardware*, de forma que os requisitos iniciais fossem atendidos e fossem plausíveis de desenvolvimento ou aquisição, dentro do cronograma do projeto. Outra dificuldade encontrada foi contornar as limitações do dispositivo *Honeywell* em relação à sua estabilidade e facilidade de programação. Além disso, as considerações sobre segurança e indisponibilidade na Internet influenciaram consideravelmente no nível de complexidade, porém mostraram-se de extrema importância para o conceito do trabalho.

Por fim, o ponto vulnerável do sistema é o fato de um crachá, conforme sistemática implementada pela UTFPR, nunca poder ser cancelado. Ou seja, ao solicitar uma segunda via de um crachá extraviado, por exemplo, o código do crachá é o mesmo do crachá antigo, implicando em um menor nível de segurança. Ao final do trabalho, percebeu-se que a fonte de tensão escolhida não fornece uma carga adequada à bateria.

Como sugestão para projetos que venham a ser desenvolvidos nessa área pode-se citar: a utilização do RFID como método de identificação no lugar do módulo *Honeywell*; a expansão do conceito para provedores com IP dinâmico; o registro de presença de alunos; assim como a modelagem de um novo sistema com escopo de aplicação em ambientes diferenciados. Além disso, a carga da bateria de 12 V poderia ser melhorada utilizando-se uma fonte de tensão maior, porém o sistema teria que ser adequado para ser alimentado nesta tensão superior.

REFERÊNCIAS

BONDI, André B. **Characteristics of scalability and their impact on performance**, In: Proceedings of the 2nd international workshop on Software and performance. Ottawa, Ontário, Canadá, 2000. p. 195 – 203. Disponível em: <http://www.win.tue.nl/~johanl/educ/2II45/2010/Lit/Scalability-bondi%202000.pdf>. Acesso em: 08 set. 2014.

BRITANNICA ACADEMIC. **Encyclopaedia Britannica Online.** Academic Edition.

CADENHEAD, Rogers. **Aprenda Java em 21 dias**. 4ªedição. Editora: Campus, Rj. 2005.

CFTV CENTER. **Catraca 4660008 ATZ 200.** Descrição do produto, 2015. Disponível em: <https://www.cftvcenter.com.br/automatizadores/automatiza/catraca-4660008-atz-200-c-netcontrol-proximidade-e-braco-que-cai-automatiza.html>. Acesso em: 09 fev. 2015.

CHOU, Jui-Sheng. Web-based CBR system applied to early cost budgeting for pavement maintenance project. **ScienceDirect**, 2008. Elsevier. Disponível em: <http://www.sciencedirect.com/science/article/pii/S095741740800047X>. Acesso em: 25 jul. 2014.

CRUZ, Renan P. **Desenvolvimento de um Sistema de Supervisão Via Web Aplicada à Automação Residencial.** 2009. Trabalho de Conclusão de Curso de Engenharia da Computação e Automação. Universidade Federal do Rio Grande do Norte, Natal, 2009.

ELEMENT 14. **RASPBERRY PI** . Disponível em: <element14.com>. Acesso em: 09 abr. 2014.

FOWLER. M, SCOTT. K. **UML essencial:** um breve guia para a linguagem – padrão de modelagem de objetos. São Paulo. Editora: Bookman, 2000.

GUEDES, G. **UML:** uma abordagem prática. São Paulo. Editora: Novatec, 2004.

HEATH, S. **Embedded Systems Design.** Burlington MA. Editora: Newnes, 2003.

HINTJENS,P. **AMQP – Imatix, 2008**. Disponível : <http://www.imatix.com/articles:whats-wrong-with-amqp> Acesso em 30 set. 2014

IMASTERS, 2011. **O que exatamente é Node.js.** Disponível em: <http://imasters.com.br/artigo/22016/javascript/o-que-exatamente-e-o-nodejs>. Acesso em: 27 abr. 2014.

ISAKOWITZ, Tomás; BIEBER, Michael; VITALI, Fabio. Web Information Systems. **Communications of the ACM**, vol. 41, n. 7, jul. 1998. Disponível em: <http://braender.tcnj.edu/mit400/articles/wis/p78-isakowitz.pdf>. Acesso em: 28 jul. 2014.

KIRUBASHANKAR, R.; KRISHNAMURTHY, K.; INDRA, J.; VIGNESH, B. Design and Implementation of Web Based Remote Supervisory Control and Information System. **International Journal of Soft Computing and Engineering**, vol. 1, n. 4, set. 2011. Disponível em: <http://www.ijsce.org/attachments/File/Vol-1_Issue-4/D080071411.pdf>. Acesso em: 27 jul. 2014.

LEITE, J.C. **Design Conceitual de Software.** Disponível em: <http://www.dimap.ufrn.br/~jair/ES/c5.html> Acesso em: 09 abr. 2014.

LES PUC RIO. **UML – Diagrama de Classes**. Disponível em: <http://www.les.inf.puc-rio.br/wiki/images/7/7f/Aula1-diagrama_classes.pdf>. Acesso em: 16 mai. 2014.

LES PUC RIO. **UML – Diagrama de Sequência**. Disponível em: <http://www.les.inf.puc-rio.br/wiki/images/e/ef/Aula02-diagrama_sequencia.pdf>. Acesso em: 16 mai. 2014.

MARQUES, Rosana. **Conceito de automação residencial ganha espaço no Mercado.** Disponível em: <http://www.aureside.org.br/imprensa/default.asp?file=10.asp> Acesso em 23 set. 2014.

MORIMOTO, Carlos E. **Linguagens de Programação (2011).** Disponível em: <http://www.hardware.com.br/artigos/linguagens>. Acesso em: 18 mai. 2014.

MOZILLA DEVELOPER NETWORK. **JavaScript.** Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Acesso em: 31 mar. 2014.

NODEJS. **NODEJS**. Disponível em:<http://nodejs.org>. Acesso em: 02 abr. 2014.

PEIXOTO, Thiago Moratori. **Sistema de Controle de Acesso Utilizando Dispositivos Embarcados**. 2013. 45 f. Monografia (Bacharelado em Ciência da Computação) – Instituto de Ciências Exatas, Universidade Federal de Juiz de Fora, Juiz de Fora, 2013. Disponível em: <http://www.ufjf.br/nrc/files/2013/05/modelo.pdf>. Acesso em: 28 jul. 2014.

RASPBERRY PI FOUNDATION, UK. **Raspberry Pi**. Disponível em: <http://www.raspberrypi.org/faqs>. Acesso em: 31 mar. 2014.

RASPBERRY PI. **Raspberry**. Disponível em: <www.raspberrypi.org/wpcontent/uploads/2011/07/RaspiModelB.png>. Acesso em: 31 mar. 2014.

RABBITMQ. **RabbitMQ Tutorial**. Disponível em: <http://www.rabbitmq.com/tutorials/tutorial-five-python.html>. Acesso em: 29 out. 2014.

RABBITMQ. **RabbitMQ – What can RabbitMQ do for you?**. Disponível em: <http://www.rabbitmq.com/features.html>. Acesso em: 02 fev. 2015.

RICHARDS, M. **Understanding the differences between AMQP and JMS**, 2011. Disponível em: <http://www.wmrichards.com/amqp.pdf>. Acesso em: 24 mai. 2014.

TEXAS INSTRUMENTS. **LM2575 Datasheet.** Dallas, 2013. Disponível em: <http://www.ti.com/lit/ds/symlink/lm1575.pdf>. Acesso em: 20 jun. 2014.

ANEXO A – CÓDIGO FONTE

Arquivo “AccessPerf.js”

```
exports.init = function init() {
    this.locations = {};
    this.locNames = [];
    var locConf = this.cm.get('locations', []);

    for (i in locConf) {
        var loc = locConf[i];

        this.locNames.push(loc.name);
        this.locations[loc.name] = {
            desc: loc.desc,
            type: loc.type || 0,
            iodev: loc.iodev,
            lockdelay: loc.lockdelay || 500,
            autolock: loc.autolock || true,
            unlocked: false
        };

        this.logger.info("Location [%s] added to internal list", loc.name);
        this.logger.debug({location: this.locations[loc.name]}, "Location [%s]
configuration", loc.name);
    }

    this.pagerefresh = this.cm.get('pagerefresh', 30);
};

exports.stop = function stop() {
    this.logger.info("** Stopping [%s] **", this.name);

    this.emit("setIndicatorState", {name: "led", value: false});
};

exports.start = function start() {
    this.logger.info("** Starting [%s] **", this.name);

    var self = this;

    setTimeout(function() {
        self.emit("setIndicatorState", {name: "led", value: true});
    }, 1000);
};

exports.renderIndexPage = function renderIndexPage(req, res) {
    if (!this.locNames.length) {
        this.logger.error("No locations configured!");
        this.emit("error", {error: "No locations configured!", req: req, res:
res});
        return;
    }

    res.render('index', {
        title: this.locNames[0],
        description: this.locations[this.locNames[0]].desc,
        locked: !this.locations[this.locNames[0]].unlocked,
        refresh: this.pagerefresh,
        ts: new Date()
    });
};

exports.renderPasswordPage = function renderPasswordPage(req, res) {
    if (!this.locNames.length) {
        this.logger.error("No locations configured!");
```

```
        this.emit("error", {error: "No locations configured!", req: req, res:
res});
        return;
    }

    res.render('password', {
        title: this.locNames[0],
        description: this.locations[this.locNames[0]].desc,
        refresh: this.pagerefresh
    });

};

exports.processAuthentication = function processAuthentication(req, res) {
    var userinfo, authObj, allowed = false;

    //some checks
    if (!this.locNames.length) {
        this.logger.error("No locations configured!");
        this.emit("error", {error: "No locations configured!", req: req, res:
res});
        return;
    }

    if (!req || !req.body || (req.body.id && req.body.password)) {
        this.logger.error("Wrong parameters");
        this.emit("error", {error: "Wrong parameters", req: req, res: res});
        return;
    }

    //get some data
    var userid = req.query.id;
    var password = req.query.password;

    //BUGFIX - remove leading zero from barcode
    userid = userid ? userid.replace(/^0+/, "") : "";

    this.logger.info("Perform authorization on userid [%s] password [exists:%s] at
location [%s]", userid, !!password, this.locNames[0]);

    //try to authenticate user
    this.emit("authenticateUser", {
        userid: userid,
        password: password
    }, processAuthorization.bind(this, req, res));
};

/**
 * Two main cases are handled here: unlockable and normal ambients.
 *
 * * The behavior for normal ambients is to allow an user to enter when there is
 * an authorization slot for this user.
 * +----------+
 * | Authed? |
 * +----------+
 * |  false  | - null
 * |  true   | <AllowOne>
 * +----------+
 *
 * * For the unlockable ambient type, the following truth table must be followed:
 * +---------+-----------+-------------+---------+
 * | Authed? | Unlocked? | inUserList? | inTime? |
 * +---------+-----------+-------------+---------+
 * |  false  |   false   |    false    |  false  | - null
 * |  true   |   false   |    false    |  false  | --impossible--
 * |  false  |   true    |    false    |  false  | - null
 * |  true   |   true    |    false    |  false  | --impossible--
 * |  false  |   false   |    false    |  true   | - null
```

```
 * |  true   |   false   |    false    |  true   | --impossible--
 * |  false  |   true    |    false    |  true   | - null
 * |  true   |   true    |    false    |  true   | --impossible--
 * |  false  |   false   |    true     |  false  | <AllowOne>
 * |  true   |   false   |    true     |  false  | --impossible--
 * |  false  |   true    |    true     |  false  | --impossible--
 * |  true   |   true    |    true     |  false  | --impossible--
 * |  false  |   false   |    true     |  true   | <Unlock>
 * |  true   |   false   |    true     |  true   | <Unlock>
 * |  false  |   true    |    true     |  true   | <Lock>
 * |  true   |   true    |    true     |  true   | <Lock>
 * +---------+-----------+-------------+---------+
 * Therefore:
 *  1. Check if user is identified
 *  2. Check if there is a timeWindow
 *   2a. If yes, lock or unlock accordingly
 *   2b. If no, just allow one user
 *
 * @param {http.req} req
 * @param {http.res} res
 * @param {Object} data
 * @param {Function} cb
 * @returns {undefined}
 */
function processAuthorization(req, res, data, cb) {
    var loc = this.locations[this.locNames[0]];
    var allowed = false;

    switch (loc.type) {

        case 1: //unlockable ambient
            if (data.user) {
                allowed = true;

                if (data.timeWindow) {
                    changeLockState(this, loc, data.user.id);
                    scheduleLock(this, loc, data.timeWindow.end);
                } else {
                    allowOne(this, loc, data.user.id);
                }
            }
            break;


        default: //default ambient
            if (data.auth) {
                allowed = true;

                allowOne(this, loc, data.user.id);
            }
    }

    var renderScanObj = {
        title: this.locNames[0],
        description: loc.desc,
        refresh: this.pagerefresh,
        allowed: allowed,
        locked: !loc.unlocked,
        userid: data.user ? data.user.id : "",
        username: data.user ? data.user.name : ""
    };

    this.logger.debug(renderScanObj, "Render scan page");
    res.render('scan', renderScanObj);
}

function allowOne(self, loc, userid) {
    self.logger.info("Ambient unlocked for user [%s]", userid);
```

```
    self.emit("setLockState", {
        name: loc.iodev,
        value: true
    });
    self.emit("allowOne", {
        event: "allowOne",
        user: userid
    });
    setTimeout(self.emit.bind(self,
            "setLockState", {
                name: loc.iodev,
                value: false
            }), loc.lockdelay);
}

function changeLockState(self, loc, userid, ensureLock) {
    loc.unlocked = ensureLock ? false : !loc.unlocked;

    self.logger.info("Ambient lock state changed to [%s] by user [%s]",
loc.unlocked, userid);

    self.emit("setLockState", {
        name: loc.iodev,
        value: loc.unlocked
    });
    self.emit("changeLockState", {
        event: "changeLockState",
        state: loc.unlocked,
        user: userid
    });
}

function scheduleLock(self, loc, end) {
    var date = new Date();
    var timeout = date.getHours() * 3600 + date.getMinutes() * 60 +
date.getSeconds();
    timeout -= end;

    if (timeout > 0) {
        setTimeout(changeLockState(self, loc, null, true), timeout * 1000);
    } else {
        setTimeout(changeLockState(self, loc, null, true), loc.lockdelay);
//immediately
    }
}
```

Arquivo “IOManager.js”

```
var util = require('util');
var events = require('events');
var fs = require('fs');

var RASPBERRY_PIN_MAPPING = {
    "3": 0,
    "5": 1,
    "7": 4,
    "8": 14,
    "10": 15,
    "11": 17,
    "12": 18,
    "13": 21,
    "15": 22,
    "16": 23,
    "18": 24,
    "19": 10,
```

```
    "21": 9,
    "22": 25,
    "23": 11,
    "24": 8,
    "26": 7
};

exports.init = function init() {
    this.devtype = this.cm.get('devtype', 'raspi-gpio');
    this.logger.info("Using [%s] as device type", this.devtype);

    if (this.devtype === 'raspi-gpio') {
        try {
            //determine Raspberry Pi Revision
(http://elinux.org/RPi_HardwareHistory#Board_Revision_History)
            this.devrev =
parseInt(fs.readFileSync("/proc/cpuinfo").toString().split("\n").filter(function(li
ne) {
                return line.indexOf("Revision") == 0;
            })[0].split(":")[1].trim(), 16) < 3 ? 1 : 2;
        } catch (e) {
            this.logger.warn({exception: e}, "Unable to determine Raspberry Pi
Board revision - assuming rev2");
            this.devrev = 2;
        }

        if (this.devrev === 2) {
            RASPBERRY_PIN_MAPPING["3"] = 2;
            RASPBERRY_PIN_MAPPING["5"] = 3;
            RASPBERRY_PIN_MAPPING["13"] = 27;
        }

        this.gpio = require('gpio');
        this.ios = {};
    }
};

exports.start = function start() {
    this.logger.info("** Starting [%s] **", this.name);

    if (this.devtype === 'raspi-gpio') {
        var ioconfig = this.cm.get('ioconfig', []);

        for (i in ioconfig) {
            if (ioconfig[i].name && ioconfig[i].addr && ioconfig[i].direction) {
                configRasPiIO(this, ioconfig[i]);
            } else {
                this.logger.error({io: ioconfig[i]}, "Failed to configure IO port,
either address or direction not configured");
            }
        }
    }
};

exports.stop = function stop() {
    this.logger.info("** Stopping [%s] **", this.name);
    var self = this;

    this.logger.debug(this.ios);
    util.puts(this.ios);

    Object.keys(this.ios).forEach(function(key) {
        self.logger.info("Unconfiguring IO [%s]:[%s]",
                self.ios[key].name, self.ios[key].addr);
        self.gpio.unexport(RASPBERRY_PIN_MAPPING[self.ios[key].addr]);
    });
};
```

```
// User Prototypes ------------------------------------------------------------

exports.setIO = function setIO(data, cb) {

    if (!this.ios[data.name]) {
        this.logger.error("Failed to set IO [%s] - does not exist", data.name);
        return;
    }

    if (this.devtype === 'raspi-gpio') {
        this.ios[data.name].iodev.set(data.value ? 1 : 0);
        this.logger.info("IO [%s] set to [%s]", data.name, data.value);
    } else {
        this.logger.warn("Dummy IO [%s] set as [%s]", data.name, data.value);
    }

    if (typeof cb === 'function')
        cb();
};

exports.getIO = function getIO(data, cb) {
    if (!this.ios[data.name]) {
        this.logger.error("Failed to get IO [%s] - does not exist", data.name);
        return;
    }

    this.logger.error("!!! getIO function not implemented !!!");

    if (typeof cb === 'function')
        cb();
};

// helpers --------------------------------------------------------------------

function configRasPiIO(instance, ioconf) {

    try {
        instance.logger.debug({conf: ioconf}, "Try to config Raspberry PI IO");

        //check pin validity
        if (!RASPBERRY_PIN_MAPPING[ioconf.addr])
            throw new Error("Invalid RaspberryPi GPIO PIN");

        //create IO object in instance
        instance.ios[ioconf.name] = {
            name: ioconf.name,
            addr: ioconf.addr,
            direction: ioconf.direction,
            ready: false,
            iodev: undefined
        };

        //prepare onReady function
        function _ready() {
            instance.ios[ioconf.name].ready = true;
            this.on("change", function(val) {

                val = (typeof val === "string" ? val.trim() === "1" : !!val);
//parse to boolean

                instance.emit("iochange", {
                    name: ioconf.name,
                    addr: ioconf.addr,
                    value: val
                });
                instance.logger.debug("IO [%s]:[%s] change to [%s]",
                        ioconf.name, ioconf.addr, val);
            });
```

```
            instance.logger.info("Ready configuring IO [%s]:[%s] as [%s]",
                    ioconf.name, ioconf.addr, ioconf.direction);
        }

        //export GPIO and create device object
        var iodev = instance.gpio.export(RASPBERRY_PIN_MAPPING[ioconf.addr], {
            direction: ioconf.direction,
            ready: _ready
        });

        //save created device object
        instance.ios[ioconf.name].iodev = iodev;

    } catch (e) {
        instance.logger.error({exception: e}, "Failed to configure IO on addr [%s]
and direction [%s]", ioconf.addr, ioconf.direction);
    }
}
```

Arquivo “Auther.js”

```
var crypto = require('crypto');

// User Prototypes ---------------------------------------------------------

/**
 * Authenticates and authorizes user based on stored permission table and
 * current timestamp. Emits events {allowed} or {denied} according to the
 * result, and also calls back the provided function with the result.
 *
 * @param {string} userID id of the user to be authenticated
 * @param {function} cb optional callback function
 * @returns {Boolean} if userID is authorized
 */
exports.authenticate = function authenticate(data, cb) {
    if (!data)
        return;

    var permissions = this.cm.get("permissions", {});

    var date = new Date();
    var weekday = date.getDay();
    var time = date.getHours() * 3600 + date.getMinutes() * 60 + date.getSeconds();

    var authTimeTable;

    var cbObj = {
        auth: false,
        date: date,
        user: null,
        timeWindow: null
    };

    /*
     * 1st - identify user
     */
    var user = data.userid ?
            this.getUserInfo(data.userid) :
            (data.password ? this.getUserInfoByPass(data.password) : null);

    if (user) {
        cbObj.user = user;

        this.logger.debug("Try to authenticate user [%s] at time [%s]", user.id,
date);
```

```
        //try to authenticate user
        if (permissions[user.id] && permissions[user.id][weekday]) {

            //search for allowance in the stored timetable
            authTimeTable = permissions[user.id][weekday];
            for (i in authTimeTable) {
                if (authTimeTable[i].start <= time && authTimeTable[i].end >= time)
{
                    cbObj.auth = true;
                    cbObj.timeWindow = {
                        start: authTimeTable[i].start,
                        end: authTimeTable[i].end
                    };
                    break;
                }
            }
        }

        //if not auth, try to find a time window
        if (!cbObj.auth) {
            Object.keys(permissions).some(function(elm, idx, arr) {
                if (permissions[elm][weekday]) {

                    authTimeTable = permissions[elm][weekday];
                    for (i in authTimeTable) {
                        if (authTimeTable[i].start <= time && authTimeTable[i].end
>= time) {
                            cbObj.timeWindow = {
                                start: authTimeTable[i].start,
                                end: authTimeTable[i].end
                            };
                            return true;
                        }
                    }
                }
                return false;
            }, this);

            this.emit("userDenied", cbObj);
        } else {
            this.emit("userAllowed", cbObj);
        }

    }

    //callback, if exists
    if (typeof cb === 'function')
        cb(cbObj);

    //return auth
    return cbObj;
};

exports.getUserInfo = function getUserInfo(userID, cb) {
    var userlist = this.cm.get("userlist", {});

    var cbObj;

    if (userlist[userID]) {
        cbObj = userlist[userID];
        this.emit("userInfo", cbObj);
    }

    //callback, if exists
    if (typeof cb === 'function')
        cb(cbObj);
```

```
    return cbObj;
};

exports.getUserInfoByPass = function getUserInfoByPass(pass, cb) {
    var userlist = this.cm.get("userlist", {});
    var passlist = this.cm.get("passlist", {});

    var shasum = crypto.createHash('sha256');
    shasum.update(pass, 'utf8');
    pass = shasum.digest('hex');

    var cbObj;

    if (passlist[pass] && userlist[passlist[pass]]) {
        cbObj = userlist[passlist[pass]];
        this.emit("userInfo", cbObj);
    }

    //callback, if exists
    if (typeof cb === 'function')
        cb(cbObj);

    return cbObj;
};
```

Arquivo “WebApp.js”

```
var util = require('util');
var events = require('events');
var http = require('http');
var path = require('path');

var express = require('express');

exports.init = function init() {
    var self = this;

    this.app = express();

    this.app.set('env', this.cm.get('appmode', 'production'));
    this.app.set('port', this.cm.get('port', 3000));
    this.app.set('views', path.join(__dirname, this.cm.get('views.path',
'views')));
    this.app.set('view engine', this.cm.get('views.engine', 'jade'));
    this.app.use(express.logger({
        format: this.app.get('env') === 'development' ? 'dev' : 'default',
        stream: {
            write: function(str) {
                self.logger.debug(str.trim());
            }
        }
    }));
    this.app.use(express.json());
    this.app.use(express.urlencoded());
    this.app.use(express.methodOverride());
    this.app.use(this.app.router);
    this.app.use(express.static(path.join(__dirname, this.cm.get('staticContent',
'public'))));

    // development only
    if (this.app.get('env') === 'development') {
        this.app.use(express.errorHandler());
    }

    //  page mappings
    this.mappings = this.cm.get("pagemappings", []);
```

```
    var map;
    for (i in this.mappings) {
        map = this.mappings[i];
        if (map.router) {
            try {
                this.app[map.method](map.url,
require(map.router)[map.function].bind(this)); //bind here?
                this.logger.info("Mapped URL [%s] with method [%s] to router [%s]
on function [%s]", map.url, map.method, map.router, map.function);
            } catch (e) {
                this.logger.error({exception: e}, "Failed to map URL [%s] with
method [%s] to router [%s] on function [%s]", map.url, map.method, map.router,
map.function);
            }
        } else {
            try {
                if (!map.function)
                    map.function = "processRequest";

                this.app[map.method](map.url, this[map.function].bind(this));
                this.logger.info("Mapped URL [%s] with method [%s] on self function
[%s]", map.url, map.method, map.function);
            } catch (e) {
                this.logger.error({exception: e}, "Failed to map URL [%s] with
method [%s] on self function [%s]", map.url, map.method, map.function);
            }
        }
    }
};

exports.start = function start() {
    this.logger.info("** Starting [%s] **", this.name);

    var self = this;

    try {
        // start app
        http.createServer(this.app).listen(this.app.get('port'), function() {
            self.logger.info('WebApp server listening on port [%d] on [%s]
environment', self.app.get('port'), self.app.get('env'));
        });
    } catch (e) {
        this.logger.error({exception: e}, "Failed to start WebServer");
    }
};

exports.getMappings = function getMappings() {
    return this.mappings;
};

exports.processRequest = function processRequest(req, res) {
    this.logger.info("New request on [%s]", req.url);
    //this.logger.debug({req: req, res: res}, "Request content");
    this.emit("pageRequest", {req: req, res: res});
};

exports.renderError = function renderError(err, req, res) {
    this.logger.info({error: err}, "Rendering error page");

    if (res) {
        res.send(500, "<b>¡Ay, caramba!</b><br>Something went wrong. Check logs for
more infos<br>" + err.toString());
    }
};
```

Arquivo “MsgManager.js”

```
var util = require('util');
var events = require('events');
var fs = require('fs');
var amqp = require('amqplib');

var DEFAULT_URL = "amqp://localhost";

exports.init = function init() {
    var cacertfiles, i;

    //get url
    this.url = this.cm.get("url", DEFAULT_URL);

    //load certificates
    cacertfiles = this.cm.get("cacerftiles", []);
    this.cacert = [];

    for (i in cacertfiles) {
        try {
            this.cacert.push(fs.readFileSync(cacertfiles[i]));
        } catch (e) {
            this.logger.warn("Could not load certificate file [%s]",
cacertfiles[i]);
        }
    }
};

exports.start = function start() {
    this.logger.info("** Starting [%s] **", this.name);

    var self = this;
    amqp.connect(this.url, {ca: this.cacert}).then(onConnect.bind(self),
function(err) {
        this.logger.error("Failed to connect to AMQP broker", err);
    });

};

exports.stop = function stop() {
    this.logger.info("** Stopping [%s] **", this.name);

    this.channel.close();
    this.conn.close();
    this.logger.info("Connection to [%s] closed **", this.url);
};

exports.handleCommand = function handleCommand(msg) {
    if (!msg || !msg.content)
        return;

    var msgstring = msg.content.toString();
    var message;

    this.logger.debug("Command message received [%s]", msgstring);

    try {
        message = JSON.parse(msgstring);
    } catch (e) {
        this.logger.error("Invalid JSON message received: [%s]", msgstring);
        return;
    }

    this.logger.info("Received command [%s]", message.command);
    this.emit("_command", message);
    this.emit("#" + message.command, message.data);

    this.channel.ack(msg);
```

```
};

exports.handleConfig = function handleConfig(msg) {
    if (!msg || !msg.content)
        return;

    var msgstring = msg.content.toString();
    var message;

    this.logger.debug("Config message received [%s]", msgstring);

    try {
        message = JSON.parse(msgstring);
    } catch (e) {
        this.logger.error("Invalid JSON message received: [%s]", msgstring);
        return;
    }

    this.logger.info("Received config for module [%s], reload [%s]",
message.module, message.reload);
    this.emit("_config", message);
    this.emit("_config_" + message.module, message);

    this.channel.ack(msg);
};

exports.publishCommand = function publishCommand(data, cb) {

    var destination = data.destination,
            command = data.command,
            msgData = data.data;

    if (!command)
        return;

    var ts = new Date().toJSON();
    var msg = {
        command: command.toString(),
        timestamp: ts,
        data: msgData
    };
    var key = destination.toString() + ".cmd." + command.toString();

    publishMessage.call(this, "dmcs.cmd", key, msg);
};

exports.publishEvent = function publishEvent(data, cb) {
    var event = data.event || data.name;

    if (!event)
        return;

    var ts = new Date().toJSON();
    var msg = {
        uuid: this.uuid,
        timestamp: ts,
        event: event.toString(),
        data: data
    };
    var key = "evt." + event.toString();

    publishMessage.call(this, "dmcs.evt", key, msg);
};

// Helper functions ------------------------------------------------------------

function generateUUID() {
    var d = new Date().getTime();
```

```
    var uuid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c)
{
        var r = (d + Math.random() * 16) % 16 | 0;
        d = Math.floor(d / 16);
        return (c === 'x' ? r : (r & 0x7 | 0x8)).toString(16);
    });
    return uuid;
}

function onConnect(conn) {
    var self = this;
    this.logger.info("Connection to [%s] established", this.url);
    this.conn = conn;
    this.conn.createChannel().then(onChannelCreated.bind(self), function(err) {
        this.logger.error("Failed to create channel", err);
    });
}

function onChannelCreated(ch) {
    this.channel = ch;
    this.logger.info("Channel [%d] created", this.channel.ch);

    initChannel.call(this);
}

function publishMessage(exchange, key, msg) {
    if (!this.channel) {
        this.logger.error("There's no available channel to send the message [%s]",
key);
        return;
    }

    this.logger.debug("Preparing message [%j] to send with key [%s] to [%s]", msg,
key, exchange);
    this.channel.publish(exchange, key, new Buffer(JSON.stringify(msg)));
    this.logger.info("Message [%s] sent to [%s]", key, exchange);
}

function initChannel() {
    var self = this;
    var newdev;

    this.uuid = this.cm.get("uuid");
    if (this.uuid === undefined) {
        //we don't know who we are. Let's define it!
        newdev = true;
        this.uuid = generateUUID();
        this.cm.set("uuid", this.uuid);
        this.cm.persistConfiguration();
    }

    var exchange_opts = {
        durable: true,
        internal: false,
        autoDelete: false
    };

    var sCmdQueue = "cmd_" + this.uuid;
    var sCfgQueue = "cfg_" + this.uuid;

    this.channel.assertExchange("dmcs.cmd", "topic", exchange_opts)
            .then(this.channel.assertExchange("dmcs.cfg", "topic", exchange_opts))
            .then(this.channel.assertExchange("dmcs.evt", "topic", exchange_opts))
            .then(this.channel.assertQueue(sCmdQueue))
            .then(this.channel.assertQueue(sCfgQueue))
            .then(this.channel.bindQueue(sCmdQueue, "dmcs.cmd", self.uuid + ".#"))
            .then(this.channel.bindQueue(sCfgQueue, "dmcs.cfg", self.uuid + ".#"))
            .then(this.channel.bindQueue(sCmdQueue, "dmcs.cmd", "_all.#"))
```

```
            .then(this.channel.consume(sCmdQueue, self.handleCommand.bind(self)))
            .then(this.channel.consume(sCfgQueue, self.handleConfig.bind(self)))
//*/
            .then(function() {
                if (newdev) {
                    //If we're a new device, let's introduce ourself!
                    self.publishCommand("", "newdevice", {uuid: self.uuid});
                }
            })
            .then(function() {
                self.logger.info("Channel initialized successfully");
            }, function(err) {
                self.logger.error("Fail to initialize channel", err);
            });
}
```