UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

AUTOMAÇÃO DE RESIDÊNCIAS ATRAVÉS DE APLICAÇÃO INTEGRADA COM ARDUINO

DANIEL PONICK BOTKE

BLUMENAU
2014

2014/2-04

DANIEL PONICK BOTKE

AUTOMAÇÃO DE RESIDÊNCIAS ATRAVÉS DE APLICAÇÃO INTEGRADA COM ARDUINO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação— Bacharelado.

Prof. Miguel Alexandre Wisintainer, Mestre - Orientador

BLUMENAU
2014

2014/2-04

AUTOMAÇÃO DE RESIDÊNCIAS ATRAVÉS DE APLICAÇÃO INTEGRADA COM ARDUINO

Por

DANIEL PONICK BOTKE

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Miguel Alexandre Wisintainer, Mestre – Orientador, FURB
Membro:	Prof. Antônio Carlos Tavares, Mestre – FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB

Blumenau, 03 de dezembro de 2014.

Dedico este trabalho primeiramente a Deus que em todo tempo me ajudou e me deu força e sabedoria para continuar, à minha noiva que esteve sempre ao meu lado me apoiando e à minha família que desde sempre tem sido meu suporte.

AGRADECIMENTOS

A Deus, pelo seu imenso amor, graça, misericórdia e força.

À minha noiva, Giovana de Sio Puetter, que não desistiu de me apoiar e de acreditar em mim mesmo nos momentos mais conturbados.

À minha família que me apoiou incessantemente durante todo este processo.

Aos meus amigos, pelos incentivos. Especialmente a Tuane Brüning Wessler, Renan Moritz, e Jonatan de Freitas pelas boas risadas e parcerias durante estes quatro anos.

Ao meu orientador, professor Miguel Alexandre Wisintainer, por ter acreditado na conclusão deste trabalho.

Aos professores do Departamento de Sistemas e Computação da Universidade Regional de Blumenau por suas contribuições durante os semestres letivos.

A vida começa com uma palmada para que comecemos a chorar. Existem pessoas, que precisam de mais uma para entender que a vida já começou.

Daniel Ponick Botke

RESUMO

Diante do cotidiano e da rotina atribulada nos dias de hoje, as pessoas cada vez têm menos tempo para dedicar às atividades de lazer, à família. Com o objetivo de reduzir este tempo despendido nas atividades do dia a dia em uma casa, foram sendo criados ao longo do tempo, vários projetos e até utopias de residências automatizadas, ou também chamadas de inteligentes. Este trabalho apresenta uma aplicação que permite a automatização de uma parte de uma residência, podendo controlar alguns equipamentos a partir de dispositivos computacionais. Para o desenvolvimento utilizou-se a placa microcontroladora Arduino em conjunto com a *shield Ethernet* para o controle dos equipamentos da residência, e o *framework* JavaServer Faces para o desenvolvimento da aplicação *web*, em conjunto com as bibliotecas Social Auth, Quartz, Primefaces, Hibernate e como banco de dados MySQL. Os resultados obtidos permitem inferir que a tecnologia adotada foi adequada aos propósitos estabelecidos.

Palavras-chave: Automação de residências. Arduino. Java Server Faces.

ABSTRACT

Due the daily and routine increasingly busy these days, fewer has time to devote to leisure activities, family, and other things besides the activities of everyday life. With the objective to reduce this time spent in the day to day in a house, was being created over time, various projects and even utopias of automated homes, or also known as smart houses. In order to promote this improvement in the quality of life of society, this paper presents an application to the automation of a part of a house, being able to control some equipment from computing devices. For the development was used the microcontroller Arduino with the Ethernet shield for the control of house equipment, and JavaServer Faces for the development of the web application, together with the Social Auth libraries, Quartz, Primefaces, Hibernate and as MySQL database. The results obtained with the construction of residential automation application influence positively the daily lives of users, automating actions, thereby assisting in the daily activities of the user, facilitating the monitoring and control of the residence.

Keywords: Residential automation. Arduino. Java Server Faces.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

DSC – Departamento de Sistemas e Computação

EEPROM – *Electrically-Erasable Programmable Read-Only Memory*

FTP – *File Transfer Protocol*

HTTP – *Hypertext Transfer Protocol*

IDE – *Integrated Development Environment*

IP – *Internet Protocol*

OSI – *Open Systems Interconnection*

SMTP – *Simple Mail Transfer Protocol*

SQL – *Structured Query Language*

SRAM – *Static Random Access Memory*

TCP – *Transmission Control Protocol*

UDP – *User Datagram Protocol*

URL – *Uniform Resource Locator*

USB – *Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

Há décadas atrás, pouco se podia pensar ou imaginar no que se tange à automação como esta é vista e aplicada nos dias atuais. Nos anos 70, segundo Schnitz e Carvalho (1988, p. 14, grifo do autor) "O *modo central de organização do trabalho*, fundado no taylorismo (quaisquer que sejam suas formas e dimensões variantes), atinge os limites de sua eficácia, ou, para colocar claramente, entra em crise".

Simultaneamente a esta crise produtiva, iniciou-se a crise econômica, o que influenciou e determinou a direção das inovações tecnológicas, mesmo que essas já viessem sendo pensadas e aplicadas nas fábricas. A partir disto tem-se um impulso e, consequentemente, um crescimento da automação, principalmente no cenário manufatureiro (SCHNITZ; CARVALHO, 1988, p. 14).

Em razão disto, segundo Schnitz e Carvalho (1988, p. 21) "...ao longo dos anos 70 assiste-se ao surgimento de uma nova geração de equipamentos no mercado que abre à automação domínios consideravelmente ampliados e múltiplos de aplicação". Porém, apenas nos anos 90 a tecnologia passou a possibilitar novas formas de promover o desenvolvimento de automações no âmbito social, a um custo acessível (BOLZANI, 2004, p. 1).

Com isso, abriram-se várias oportunidades e um grande mercado para a "indústria da automação" no âmbito social. Segundo Bolzani (2004, p. 1) "... neste panorama, não se pode ser de forma alguma um ludista[1], contrário à evolução da tecnologia, mas utilizá-la, de forma mais veemente, para solução de problemas cotidianos".

Nos dias atuais, existe uma lacuna do mercado de automação, sendo esta a de automação residencial, que está pouco preenchida no cenário nacional. Além disto, das companhias existentes neste nicho de mercado, poucas, ou quase nenhuma, oferecem a solução a um preço acessível para grande maioria da população.

O preço elevado destas soluções dá-se, em sua maioria, devido a não existência de um sistema único para gerenciamento, independente de outros fornecedores. A maior parte das empresas de automação residencial fazem pacotes de equipamentos de automação de outros fornecedores, um agregado ao outro, o que aumenta o custo final de uma residência automatizada, e pode diminuir também os ganhos proporcionados por esta.

---

[1] Ned Ludd foi um trabalhador inglês que teria destruído teares em 1779, sendo contra a mecanização. Por extensão, um ludista é qualquer um que se oponha a avanços técnicos ou tecnológicos.

Frente a isto, poucos são os sistemas de automação residencial que podem ser adquiridos por boa parte da população brasileira, considerando o poder aquisitivo médio brasileiro. Isto faz com que a maioria da população não possa usufruir do conforto, economia de tempo e maior qualidade de vida proporcionados por esses sistemas.

Embora existam muitas soluções de automação residencial, mundialmente falando, nacionalmente pouco se explora esse mercado, principalmente no que tange à uma solução de baixo custo. Tem-se a partir disto um mercado aberto e não explorado de automação residencial para consumidores com menor poder aquisitivo.

1.1 OBJETIVOS DO TRABALHO

O objetivo geral do trabalho é desenvolver uma aplicação para controle de residências que facilite a execução das atividades cotidianas em uam residência.

Os objetivos específicos do trabalho são:

a) desenvolver uma central de controle residencial baseada em Arduino que se integrará com a aplicação;
b) permitir o controle de iluminação residencial;
c) permitir o controle da climatização residencial;
d) permitir o monitoramento da situação das janelas;
e) identificar possíveis padrões e sugerir a realização de tarefas automaticamente;

1.2 ESTRUTURA DO TRABALHO

No primeiro capítulo tem-se a introdução ao tema principal deste trabalho com a apresentação da justificativa e dos objetivos.

No segundo capítulo apresentam-se a fundamentação teórica pesquisada sobre automação, automação residencial, microcontroladores, Arduino, mineração de dados, além de trabalhos correlatos.

O terceiro capítulo apresenta o desenvolvimento da aplicação iniciando-se com o levantamento de informações, tendo na sequência a especificação com os requisitos da

aplicação, diagramas de caso de uso e modelo de entidade relacionamento, em seguida a descrição da implementação com técnicas e ferramentas e a operacionalidade da ferramenta, e por fim os resultados e discussão do trabalho.

No quarto capítulo tem-se as conclusões deste trabalho bem como apresentam-se sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda assuntos a serem apresentados nas seções a seguir, tais como automação, automação residencial, microcontroladores, Arduino, mineração de dados, além de trabalhos correlatos.

2.1 AUTOMAÇÃO

Pode-se resumir como automação tudo aquilo que uma máquina programada tem capacidade de fazer, repetidas vezes, no lugar de um ser humano. Tem-se como finalidade o aumento da produtividade e a redução dos custos de uma companhia.

> A transformação concreta de processos de trabalho baseados em técnicas artesanais em modo especificamente capitalista de produção, implicou (e ainda implica) em materialização sistemática de habilidades, conhecimento e informações ligadas à produção – anteriormente sob o domínio de trabalhadores – em equipamentos de propriedade do capital. (SCHNITZ; CARVALHO, 1988, p. 65).

Na década de 80, a automatização se impregnou na indústria e desde lá vem modificando algumas características capitalistas. A partir de então, a base técnica eletromecânica se difundiu com a indústria, permitindo automatizar atividades desde que estas fossem repetitivas (SCHNITZ; CARVALHO, 1988, p. 64 - 65).

Nesta época, o grande salto de produtividade e qualidade no que se conhece por produção capitalista, foi percebido principalmente nos processos produtivos em série, que produzem produtos sempre iguais ou similares. A automatização destes processos, foi marcada e alavancada com o surgimento e uso da microeletrônica (SCHNITZ; CARVALHO, 1988, p. 65).

Essa automatização por microeletrônicos por sua vez foi impulsionada pela impressionante miniaturização e queda dos preços dos componentes eletrônicos, que vem ocorrendo continuamente. Com isto, gerou-se ganho com redução de custos da mão-de-obra, melhoria nas condições de trabalho, aumento da qualidade e flexibilidade na produção, facilidade de se inserir em um novo mercado, maior controle da produção, entre outros (SCHNITZ; CARVALHO, 1988, p. 66 - 72).

Os setores que mais experimentaram destes ganhos e foram pioneiros com vários *cases* de sucesso são o têxtil, o vestuário e confecções, as máquinas e ferramentas, a automobilística, a eletrônica e as telecomunicações. Hoje em dia, no entanto, não há setor em que não se consiga ou não se utilize de alguma forma ou grau a automatização de atividades (SCHNITZ; CARVALHO, 1988, p. 75 - 123).

A automação destas atividades, de acordo com Mecatrônica Atual (2013, p. 1) "... é resultante da integração de três tecnologias básicas: Sensores, Controladores e Atuadores". Sem a presença de uma destas tecnologias, o sistema fica incompleto e não proporciona o máximo de seu resultado. A seguir tem-se uma breve explicação de cada uma destas tecnologias:

a) sensores são os responsáveis por traduzir grandezas físicas em sinais elétricos correspondentes, como sensor de presença, luminosidade, entre outros;
b) atuadores são os componentes responsáveis por realizar ações que seriam realizadas manualmente, como uma haste pneumática, um transmissor de infravermelho, relés, entre outros;
c) controladores são os que integram os sensores e os atuadores para que o sistema atinja o seu objetivo, bem como permite a interação de usuários com o sistema.

2.1.1 AUTOMAÇÃO RESIDENCIAL

A automação residencial, também conhecida como domótica, visa a automação e o controle de toda uma residência a partir de dispositivos eletrônicos. Essa automatização se dá através de um integrador que se comunica através de protocolos com a rede física (BOLZANI, 2004, p. 51 - 53).

Essa automação tem como objetivo proporcionar conforto aos seus usuários, uma vez que ações simples como controle de iluminação, climatização, segurança, entre outros, podem ser realizadas de forma prática e rápida através do controlador do sistema. O quanto esses recursos poderão ajudar o usuário dependerá de seu estilo de vida, dos gostos pessoais e de seus recursos disponíveis (MARTE, 1995, p. 15).

Junto a estas ações disponibilizadas ao usuário, segundo Dias e Pizzolato (2004), "Dispositivos como detectores, sensores, captadores e atuadores trocam informações entre eles ou com unidades centrais inteligentes, sendo capazes de processar os dados recebidos e

enviar sinais ...". As informações repassadas por estes dispositivos podem ser utilizadas para gerar alertas, controles ou qualquer outra informação que seja relevante ao usuário.

Esses dispositivos e suas formas de uso vêm crescendo cada vez mais, com o crescente número de usuários e novos dispositivos sendo criados e ofertados. Um sistema domótico, portanto, deve ser pensado e estruturado de forma que seja possível adicionar estes recursos, funcionalidades, e integrações, sem esforço excessivo.

2.2 MICROCONTROLADORES

Os microcontroladores são computadores em um *chip*, ou um *single-chip* que além de serem pequenos (micro) possuem a capacidade de controlar objetos, processos ou eventos (controladores). Os microcontroladores também podem ser denominados de controladores embutidos, pois estes suportam os circuitos que são construídos nos dispositivos controlados (FORESTI, 2013).

O primeiro microcontrolador conhecido, foi catalogado em 1975, sendo um microcomputador capaz de se auto programar. Este microcontrolador não incluía nenhum dos periféricos ou mídias conhecidas, ele era programado por botões simples de duas posições existentes em seu painel. A grande revolução neste microcontrolador chamado Altair ocorre quando a Microsoft ofereceu um sistema de programação para ele (AXELSON, 1997, p. 2).

Hoje em dia, no entanto, a realidade dos computadores é totalmente outra, a evolução é presente no dia-a-dia. Além do aumento significativo da capacidade dos computadores, com mais memória, processamento e armazenamento, foram adicionados vários periféricos que facilitam seu uso e aumentam sua aplicabilidade.

Devido a esta evolução diária, segundo Axelson (1997, p. 1, tradução nossa) "Você pode encontrar todos os tipos de microcontroladores nos dias de hoje". Um exemplo, e o mais popular nos dias atuais, são os dispositivos móveis, que utilizam microcontroladores para realizar suas tarefas.

2.2.1 Arduino

O Arduino é uma plataforma de microcontrolador que tem como principal diferencial a sua facilidade de uso e natureza aberta. Segundo Monk (2013, p. 5) o Arduino "Basicamente, permite que você conecte circuitos eletrônicos aos seus terminais de modo que ele possa controlar dispositivo – por exemplo, ligar ou desligar lâmpadas e motores, ou medir coisas como luz e temperatura".

Para realizar estas atividades, o Arduino possui como sua peça principal, um microcontrolador de 28 pinos produzido pela Atmel, uma das maiores fabricantes de microcontroladores. Esse microcontrolador possui uma memória para receber dados e toda a eletrônica necessária para os pinos de entrada e saída (MONK, 2013, p. 6).

Esse microcontrolador, no entanto, não é o único, uma vez que existe uma grande variedade de outros microcontroladores como esse produzidos e comercializados, usados para os mais diversos fins. O que diferencia o Arduino é a sua multifuncionalidade em um único microcontrolador padrão (MONK, 2013, p. 6).

O Arduino, portanto, consiste na junção deste microcontrolador a uma placa de desenvolvimento de hardware aberto. Todos os arquivos, projetos e diagramas da placa estão disponíveis para toda e qualquer pessoa que queira produzir e comercializar sua placa Arduino (MONK, 2013, p. 7).

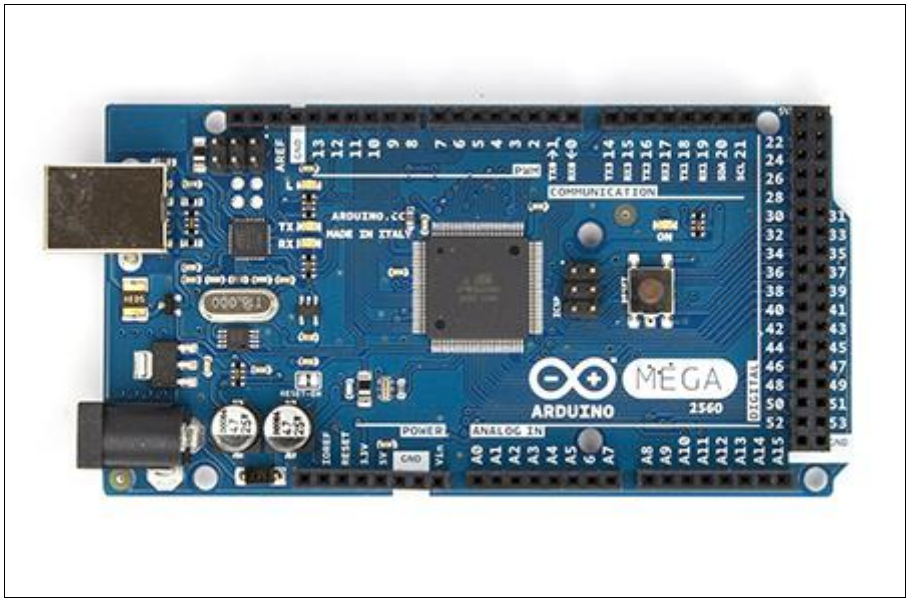
Inicialmente esta placa foi construída com o propósito de auxiliar no ensino de estudantes, no entanto em 2005 ele foi pela primeira vez desenvolvido comercialmente e desde então vem sendo bem sucedida devido a sua facilidade de uso. Além disso, por ser um projeto de código e hardware aberto, surgiram várias opções da placa a um custo menor, apenas com nome diferente, uma vez que apenas o nome Arduino é protegido (MONK, 2013, p. 11).

Uma outra vantagem proporcionada pelo Arduino é uma grande variedade de extensões, chamadas de *Shields*, que agregam funcionalidades a placa. Alguns exemplos de *Shileds* que pode-se listar são Ethernet, Motor, USB Host e Wireless SD (ARDUINO, 2014).

Além da variedade de *Shields*, existe também uma variedade de tipos de placa Arduino para os mais diversos fins, conhecida como a família Arduino. Algumas das placas existentes, sem considerar os clones e variantes, sendo estas as mais utilizadas são: Uno, Mega, Nano, Bluetooth e Lilypad (ARDUINO, 2014).

Em quase todos os tipos de Arduino, a placa contém portas digitais e analógicas que podem ser utilizadas de acordo com a necessidade. O número de portas disponíveis pode ser diferente para cada tipo de placa em questão, o Arduino Mega por exemplo, é a placa com maior número de portas, contendo 54 portas digitais e 16 analógicas disponíveis, conforme mostrado na Figura 1.

Figura 1 – Arduino Mega



Fonte: Arduino (2014).

No Quadro 1 são apresentadas as principais características do Arduino Mega.

Quadro 1 – Principais características do Artuino Mega

Microcontrolador	ATmega2560
Voltagem de operação	5V
Voltagem de entrada (recomendado)	7-12V
Voltagem de entrada (limites)	6-20V
Pinos digitais de entrada e saída	54 (15 fornecem saída PWM)
Pinos analógicos de entrada	16
Corrente para pinos de entrada e saída	40 mA
Corrente para pinos 3.3V	50 mA
Memória *Flash*	256 KB dos quais 8 KB são usados para inicialização
Static Random Access Memory (SRAM)	8 KB
Electrically-Erasable Programmable Read-Only Memory (EEPROM)	4 KB
Velocidade de *Clock*	16 MHz

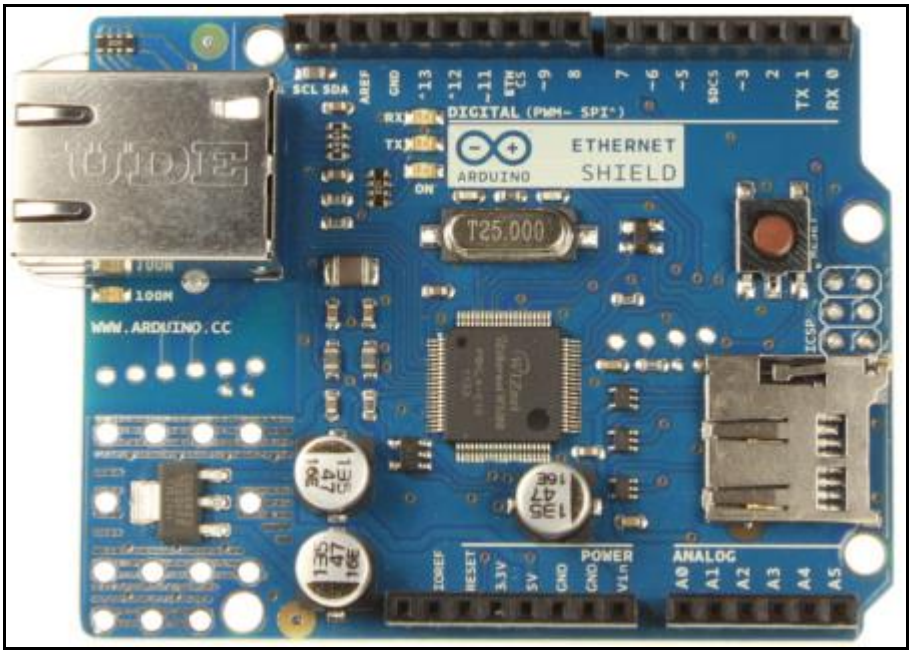
Fonte: Arduino (2014).

2.1.1.1 *Shield* Ethernet

A *shield* Ethernet é uma segunda placa que é conectada ao Arduino, permitindo que este se conecte a internet, fazendo com que o Arduino se torne um servidor *web*. Para o desenvolvimento e utilização deste servidor, é disponibilizado pelo produtor da placa uma biblioteca específica, denominada também como Ethernet.

A conexão com a internet é realizada através da conexão Rj45 disponível na *shield*. Sendo que o seu processamento é realizado através do Wiznet W5100 *ethernet chip* (ARDUINO, 2014). Na Figura 2 é apresentada a *shield* Ethernet conforme descrito.

Figura 2 – *Shield* Ethernet



Fonte: Arduino (2014).

2.3 MINERAÇÃO DE DADOS

Até alguns anos atrás, a grande preocupação da computação era o armazenamento de dados, o que ficou ainda mais evidente com a redução do custo de aquisição de hardware. Com o tempo, foram criados grandes repositórios de dados, com uma quantidade imensurável de informações (CAMILO; SILVA, 2009, p. 2).

Com o grande volume de dados produzido durante os anos e com a velocidade de atuação exigida pela economia mundial as técnicas de extração de informações tornaram-se ineficientes. Com a finalidade de atender esta nova demanda de informações, foi proposta, no final da década de 80, a mineração de dados (CAMILO; SILVA, 2009, p. 2).

A mineração de dados conta com definições de termos padrões e um processo definido, além de diferentes técnicas para a extração da informação existente em meio aos dados. Através destas técnicas, procura-se encontrar padrões existentes nos dados que possam gerar alguma informação para os usuários, que em sua maioria são os tomadores de decisão das organizações (REZENDE, 2005, p.11).

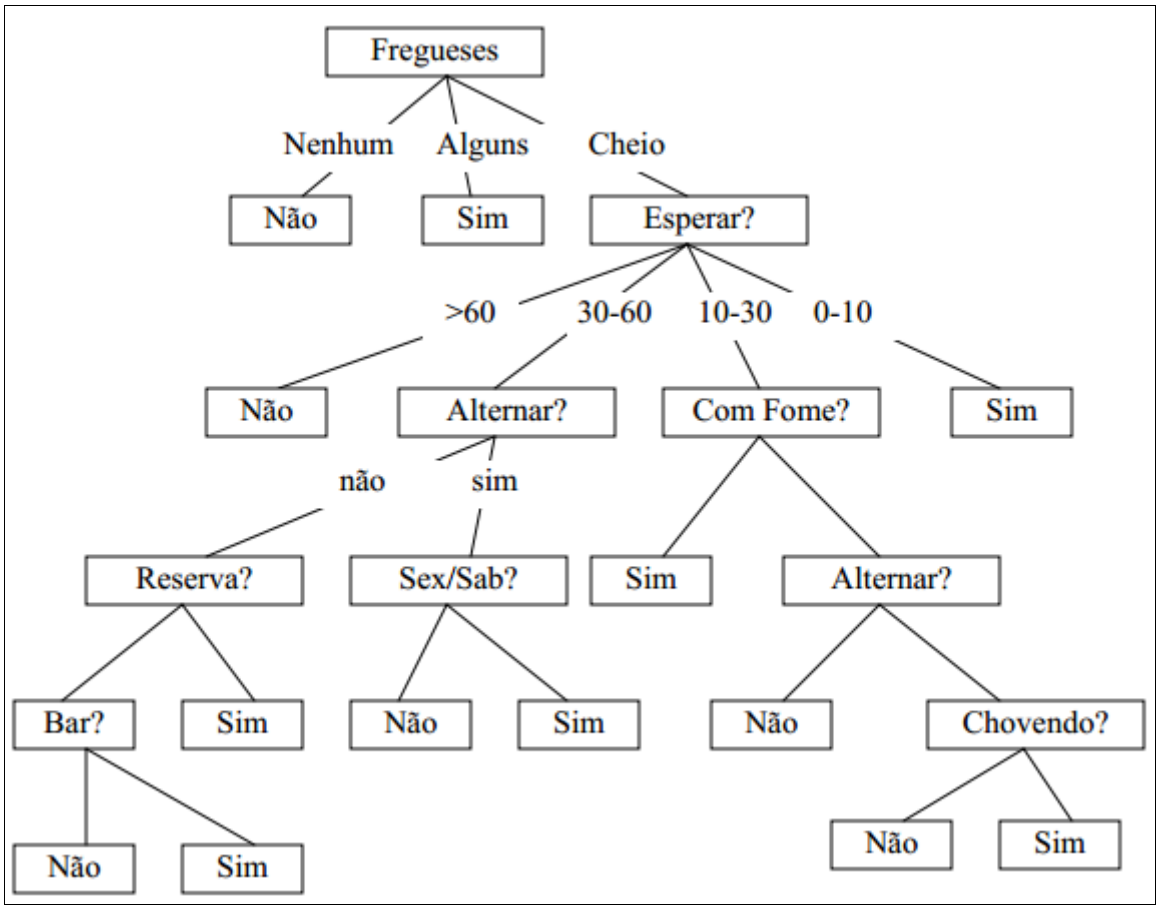
Principalmente pelo usuário, uma das preocupações da mineração de dados é a forma de apresentação e visualização das informações. Em sua grande maioria, os sistemas que aplicam a mineração de dados apresentam as informações de forma gráfica e interativa, permitindo ao usuário visualizar as informações da forma desejada e obter as informações de forma simples (REZENDE, 2005, p.21).

2.3.1 Árvore de decisão

A árvore de decisão é uma das técnicas de mineração de dados, que auxilia na tomada de decisão sobre um determinado dado. A técnica recebeu este nome pelo seu resultado e representação serem similares a uma árvore, com ramos, nós e folhas.

Para chegar a uma decisão, a técnica realiza uma sequência de testes sobre o dado, cada teste é representado por um nó. A este nó são ligados dois ramos, cada qual esperando um determinado resultado, e levando ao próximo teste, e assim sucessivamente até chegar em uma folha (nó final). Segundo Pozzer (2006) “Cada nó folha da árvore especifica o valor de retorno se a folha for atingida.” conforme apresentado na Figura 3.

Figura 3 – Modelo de representação de árvore de decisão



Fonte: Pozzer (2006).

2.4 TRABALHOS CORRELATOS

Besen (1996) desenvolveu um protótipo de controle para um cômodo de uma residência, propondo-se a controlar temperatura, luminosidade e supervisão das janelas e portas. O módulo do controlador foi desenvolvido utilizando o microcontrolador 8031, e como supervisor um computador pessoal.

Conectado ao microcontrolador, Besen (1996) utilizou-se de um sensor de temperatura somado a um sensor de contato que monitora a janela para definir quando a tomada do condicionador de ar estará ligada ou desligada. Para controlar o estado do ar condicionado foi definido que este não seria ligado com a janela aberta, e quando fechada, deveria manter a temperatura entre 15 e 28º C.

O controle da iluminação, por sua vez, resume-se a ligar ou apagar a luz de acordo com a percepção de presença no cômodo, detectada por um sensor. Esta automação tem como fim a redução do consumo desnecessário de energia.

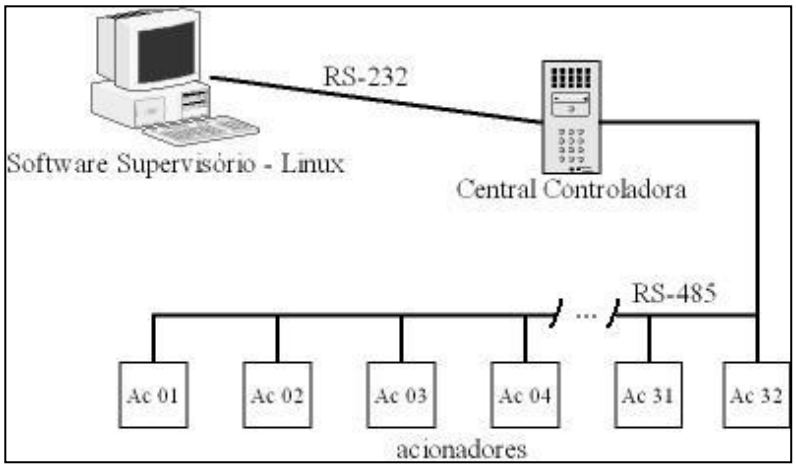
Censi (2001) desenvolveu um sistema e hardware para controle de residências, no qual o comando de fora da residência é disparado com a utilização de *emails*, e internamente utilizando esse hardware. O trabalho se propôs a controlar iluminação, condicionador de ar, alarme e eletrodomésticos. O sistema é conectado a uma caixa postal de *email*, a qual é acessada periodicamente verificando se há algum *email*. Existindo algum *email* verifica-se se o usuário do *email* recebido tem permissão para executar a tarefa e se o comando existente no corpo do *email* é válido. Caso as duas verificações sejam positivas, é registrada a tarefa recebida por *email* para ser executada no horário definido no *email*.

O *email* com o comando desejado deve seguir o padrão pré-estabelecido, contendo aparelho, horário e operação. Caso o *email* não contenha as informações descritas corretamente, ou o usuário de envio do *email* não esteja autorizado a enviar comandos a residência, o sistema envia um *email* ao usuário informando a exceção, caso contrário envia um *email* confirmando o registro da tarefa. A leitura e envio dos *email*s é realizada por um hardware baseado na placa programável Rabbit 2000 TCP/IP. Junto a placa foi adicionado um visor LCD, um controlador de relés e um teclado para que o usuário possa controlar os eletrodomésticos a partir do hardware (CENSI, 2001).

Reiter Júnior (2006) desenvolveu um sistema que permite automatizar uma residência sem a necessidade de realizar mudanças na estrutura da casa para isto. A aplicação se integra com placas acionadoras microcontroladas para a automação, estas acionadoras por sua vez são supervisionadas por um software em um computador. Dentro de cada tomada e interruptor da residência, foram colocados pequenos circuitos eletrônicos que realizam o controle dos dispositivos conectados a ele. Os dispositivos controlados são eletrodomésticos e lâmpadas. Nestes pequenos circuitos, são utilizados microcontroladores da família MCS51, interligados e respondendo a uma central controladora.

Para interação com o usuário foi desenvolvida uma aplicação para Linux, que permite o controle dos dispositivos conectados às tomadas e interruptores. A interface do sistema é expansível, permitindo ao sistema adaptar-se a qualquer residência, conforme representado na Figura 4.

Figura 4 – Disposição física do sistema

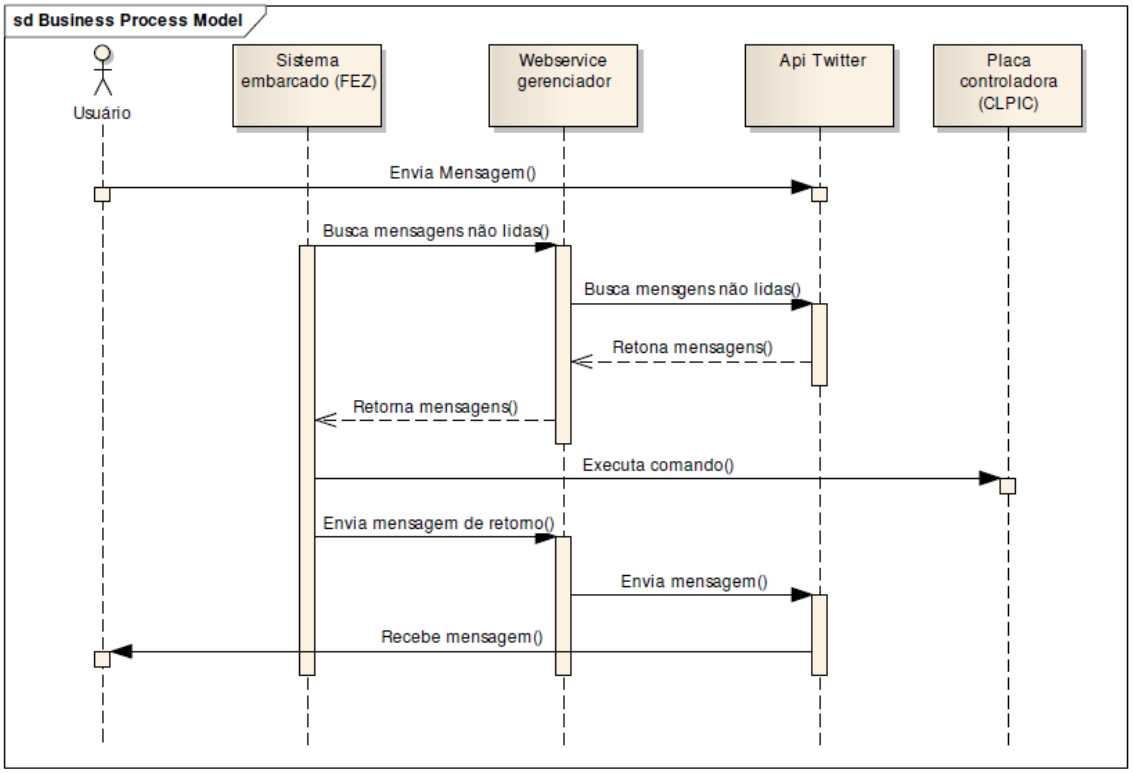


Fonte: Reiter Júnior (2006).

Gadotti (2010) desenvolveu uma aplicação para controle de residência através da rede social Twitter, que pode ser facilmente acessada de qualquer dispositivo. Com a aplicação foi possível ligar e desligar os dispositivos eletrônicos: lâmpadas, televisores, eletrodomésticos, computadores, entre outros. A configuração do sistema é realizada utilizando arquivos XML, permitindo que o administrador configure os usuários que podem controlar os eletrodomésticos, quais os eletrodomésticos que serão controlados, bem como qual a operação realizada em cada eletrodoméstico.

Tendo os usuários definidos, a aplicação verifica se há algum *tweet* do usuário que corresponde a alguma operação na residência. Sendo encontrado algum *tweet* válido, a operação é realizada através da placa microprocessadora CLPIC-628, com uma placa FEZ Domino e um *Ethernet shield*, conforme diagrama de sequência apresentado na Figura 5.

Figura 5 – Diagrama de sequência Gadotti (2010)



Fonte: Gadotti (2010).

3. DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo estão descritos os aspectos técnicos utilizados no desenvolvimento da aplicação, bem como o levantamento de informações: requisitos funcionais e não funcionais. Além dos diagramas utilizados no desenvolvimento: diagrama de casos de uso, diagrama de classes, fluxo de atividades e modelo de entidade e relacionamento, estão descritas também as técnicas e ferramentas utilizadas, a operacionalidade da implementação e os resultados obtidos.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Conforme referido anteriormente, vislumbrou-se uma oportunidade de mercado para aplicação de automação de residências que seja de baixo custo, e que permita adicionar e configurar vários dispositivos sem complicações. Esta necessidade dá-se principalmente pelo fato dos atuais fornecedores deste tipo de solução, utilizarem várias soluções diferentes integradas para fornecerem a automação de uma residência.

Com o objetivo de desenvolver uma aplicação com estas características, foi utilizado o microcontrolador Arduino, equipamento de baixo custo e de variadas possibilidades de uso. A aplicação desenvolvida possibilita o controle de iluminação e ar condicionado e o monitoramento de janelas, mas pode ser facilmente estendida para controlar outros equipamentos, uma vez que o Arduino possui uma grande combinação de funcionalidades, bem como de recursos que podem ser acoplados a ele, conforme explanado na Fundamentação teórica.

3.2 ESPECIFICAÇÃO

A seguir é apresentada a especificação da aplicação, contendo os requisitos funcionais e requisitos não funcionais, além dos diagramas de casos de uso e o modelo de entidade relacionamento (MER). Para criar o diagrama de caso de uso foi utilizada a ferramenta Enterprise

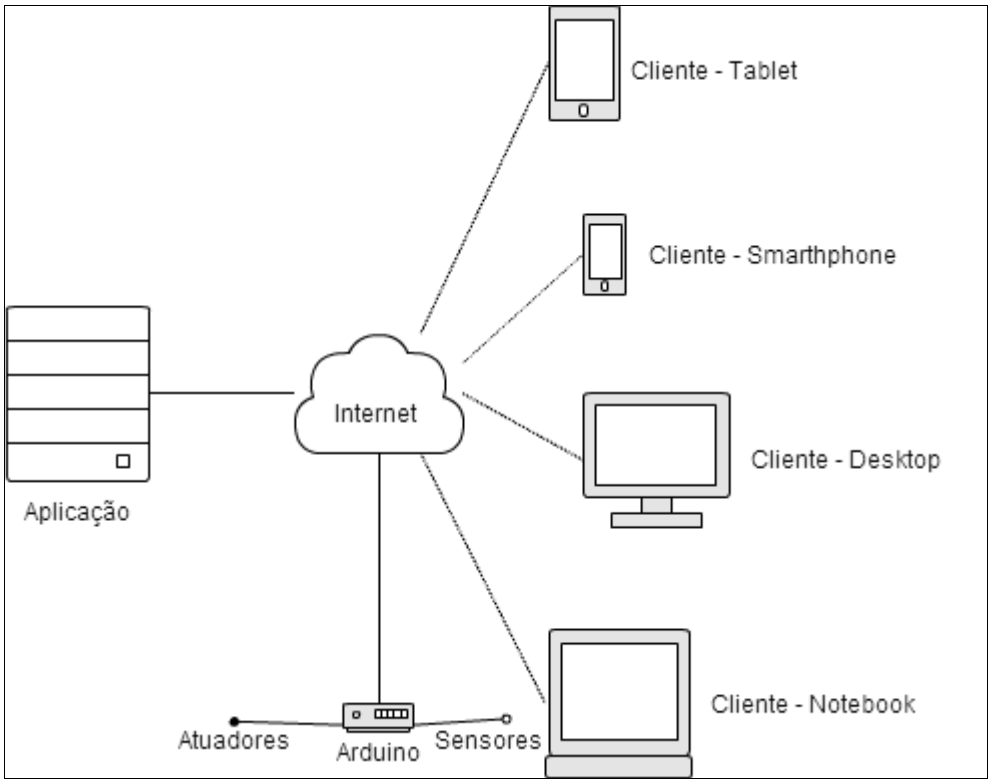
Architect (EA), já para a criação do MER utilizou-se a ferramenta MySQL Workbench.

3.2.1 Arquitetura da Aplicação

Nesta subseção são apresentadas as principais características da arquitetura da aplicação, sendo que para o desenvolvimento desta aplicação, foi utilizada a plataforma *web*, permitindo o acesso a esta por meio de qualquer dispositivo, desde computadores, até *smartphones*. Esta aplicação *web* é única para todos os usuários, sendo que o seu processamento é realizado no servidor, apresentando apenas a interface para o cliente.

Na interface apresentada, são exibidas as opções de acordo com as credenciais utilizadas e sua respectiva residência. Os Arduinos, sendo um por residência, estão ligados a aplicação por meio da internet, sendo que em cada um deles está configurada a respectiva residência, configuração esta que é utilizada para apresentar as opções para o usuário. A arquitetura desta aplicação é apresentada na Figura 6.

Figura 6 – Arquitetura da aplicação



3.2.2 Requisitos da Aplicação

Nesta subseção são apresentadas as principais características da aplicação. O Quadro 2 apresenta os requisitos funcionais previstos para a aplicação e sua rastreabilidade, ou seja, vinculação com o(s) caso(s) de uso associado(s).

Quadro 2 – Requisitos funcionais

Requisitos Funcionais	**Caso de Uso**
RF01: A aplicação deverá permitir que o usuário efetue a autenticação de acesso à aplicação.	UC01
RF02: A aplicação deverá permitir que o usuário altere sua senha.	UC02
RF03: A aplicação deverá permitir que o usuário acenda e apague as luzes de uma residência.	UC03
RF04: A aplicação deverá permitir que o usuário consulte o estado atual da luz (ligada ou apagada).	UC03
RF05: A aplicação deverá permitir que o usuário controle os condicionadores de ar da marca Komeco.	UC04
RF06: A aplicação deverá permitir que o usuário consulte a situação das janelas conectadas a central.	UC05
RF07: A aplicação deve armazenar o histórico de utilização com: ação, equipamento, data, hora, usuário.	UC06
RF08: Com a base de dados históricos do uso da aplicação, esta deve sugerir automatização de ações.	UC06
RF09: Em uma mesma tela, a aplicação deve permitir o usuário consultar todos os cômodos da residência que tem algum componente automatizado.	UC07
RF10: Em uma mesma tela, a aplicação deve permitir o usuário consultar todos as ações possíveis para um cômodo.	UC07

O Quadro 3 lista os requisitos não funcionais previstos para a aplicação.

Quadro 3 – Requisitos não funcionais

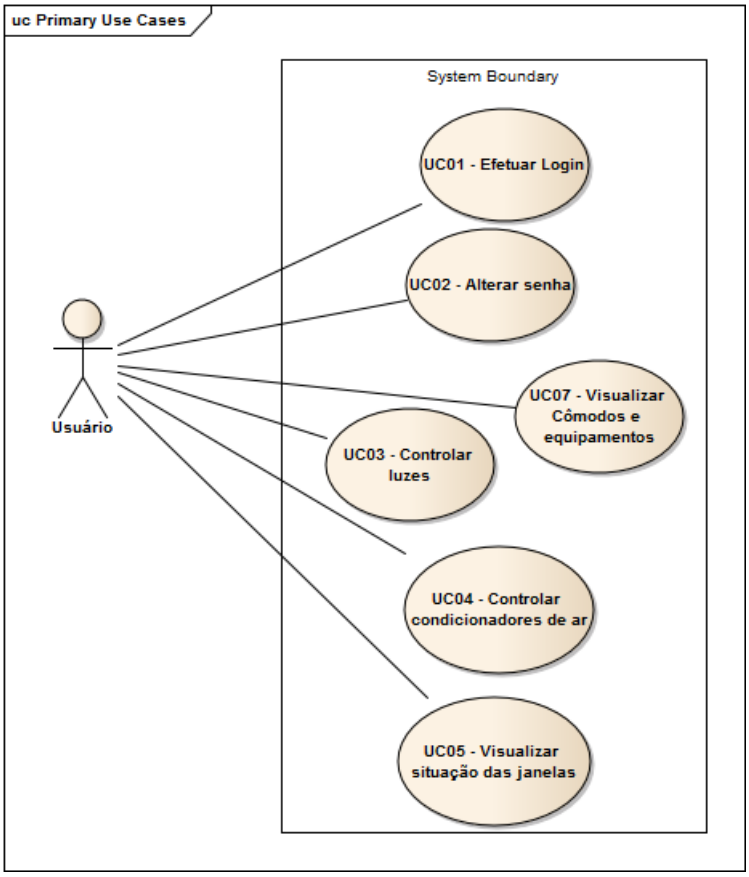
Requisitos Não Funcionais
RNF01: A aplicação deverá utilizar o protocolo HTTP para conexão com a central de controle da casa (Arduino).
RNF02: A aplicação deverá ser compatível com as plataformas Android e IOS.
RNF03: A aplicação deve utilizar o banco de dados MySQL para armazenar o histórico de uso da aplicação.
RNF04: A aplicação deve garantir a segurança dos dados de autenticação do usuário.

3.2.3 Diagrama de Caso de Uso

Esta subseção apresenta através da Figura 7 e da Figura 8, os diagramas de casos de uso da aplicação, sendo que o detalhamento dos casos de uso está descrito à partir do Apêndice A.

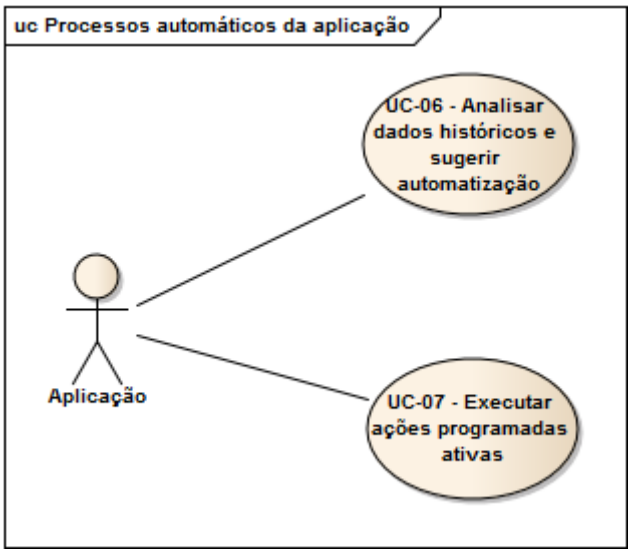
A Figura 7 apresenta os casos de uso do fluxo principal.

Figura 7 – Diagrama de caso de uso, fluxo principal



A Figura 8 apresenta os casos de uso das ações da aplicação.

Figura 8 – Diagrama de caso de uso dos processos automáticos da aplicação

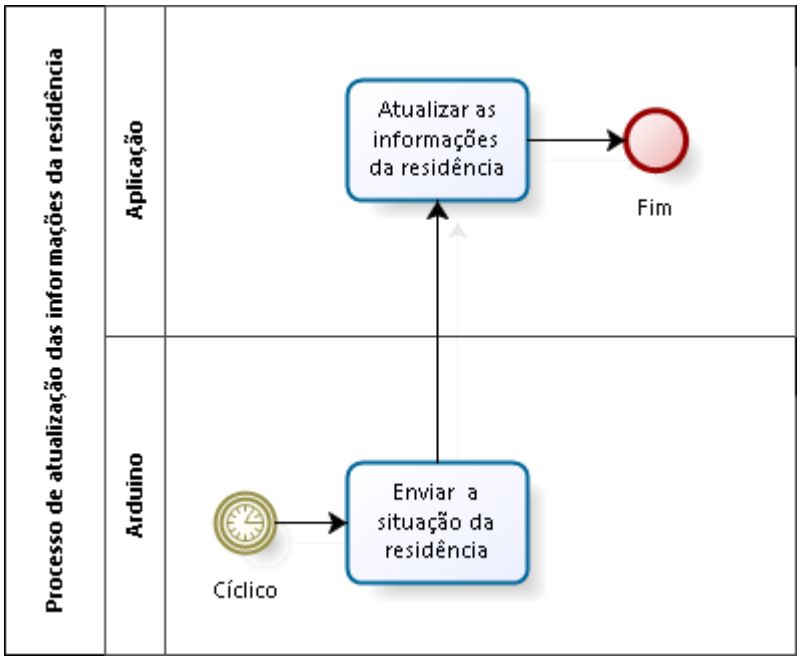


3.2.4 Fluxos de Uso e Execução da Aplicação

Nesta subseção são apresentados os fluxos de uso e execução da aplicação, tendo como base a arquitetura apresentada anteriormente. Nesta arquitetura, a aplicação retorna para o usuário a interface com os cômodos e equipamentos configurados no Arduino, sendo apresentados conforme a situação atual destes equipaments.

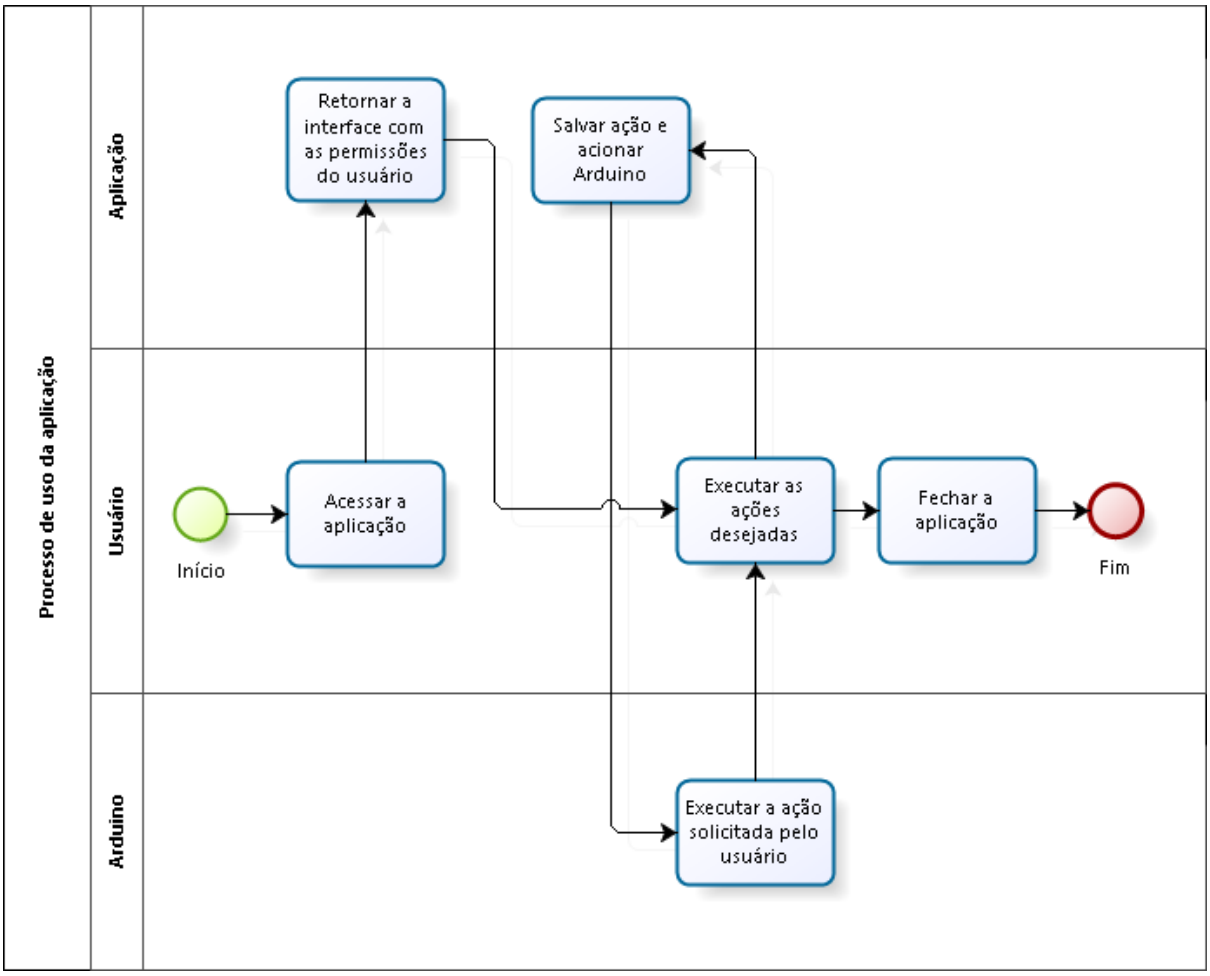
Estas informações dos equipamentos da residência (luzes e seu estado atual por exemplo) apresentadas ao usuário, são enviados pelo Arduino para a aplicação via protocolo UDP ciclicamente. Estas informações recebidas pela aplicação são utilizadas para atualizar a situação da residência na aplicação conforme a Figura 9.

Figura 9 – Processo de atualização das informações da residência



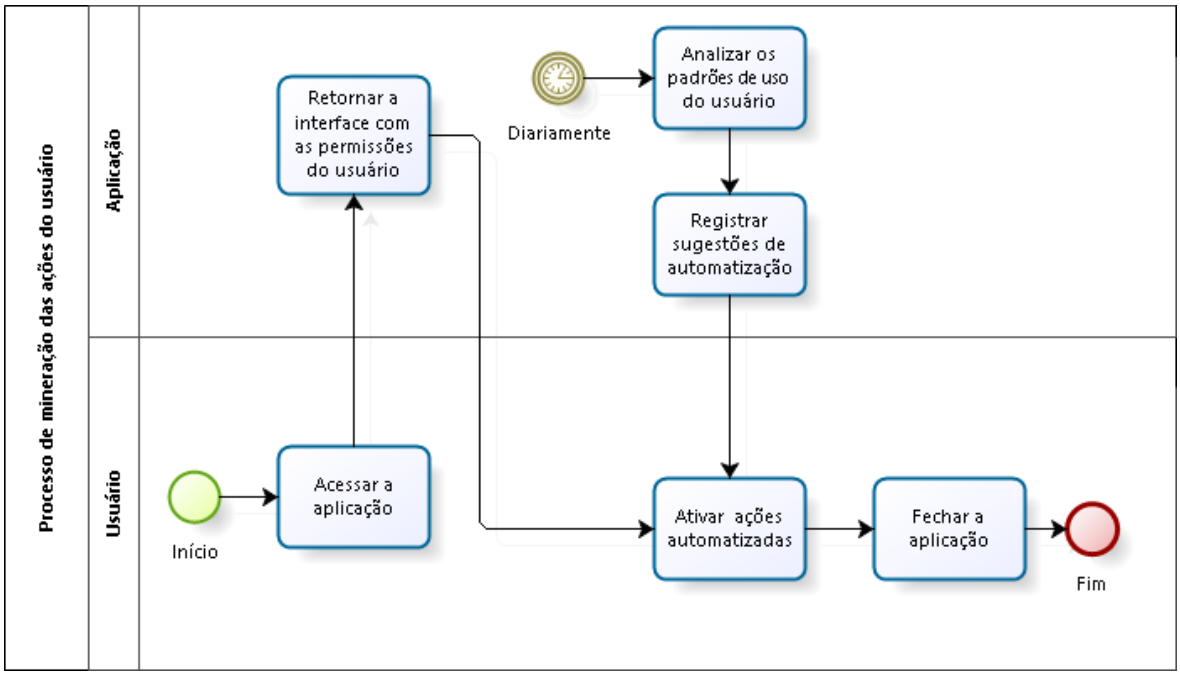
A partir do momento em que o usuário está conectado à aplicação e visualizando sua interface, todas as ações executadas por ele na residência são armazenadas no banco de dados da aplicação. Este armazenamento é realizado com o intuito de descobrir padrões e sugerir a automatização de ações repetitivas, conforme a Figura 10.

Figura 10 – Processo de uso da aplicação



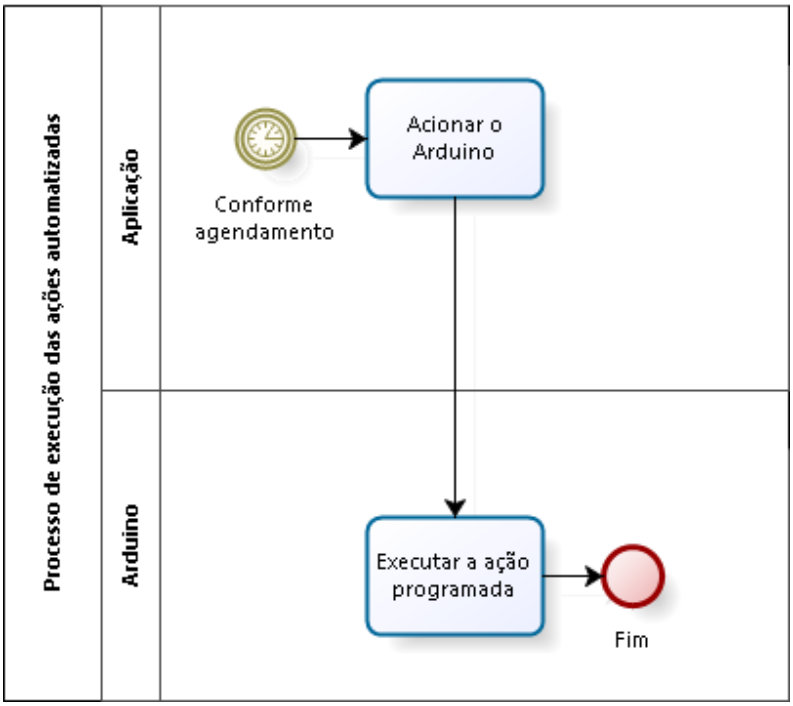
Estes padrões de uso são descobertos através da mineração das ações armazenadas, a qual é executada diariamente, em um horário de baixo fluxo de uso. A partir do momento em que um padrão de uso é encontrado, este é cadastrado como sugestão de automatização para o usuário, a partir de então, cabe ao usuário ativar ou não estas sugestões para que elas passem a ser executadas automaticamente pela aplicação na recorrência observada, conforme a Figura 11.

Figura 11 – Processo de mineração das ações do usuário



As ações ativadas pelo usuário, passam então a ser executadas conforme a recorrência em que foram programadas, de acordo com o padrão encontrado nas ações do usuário. Quando executada a ação programada, esta não é armazenada como uma ação do usuário, afim de não influenciar na identificação dos padrões de uso da aplicação conforme a Figura 12.

Figura 12 – Processo de execução das ações automatizadas

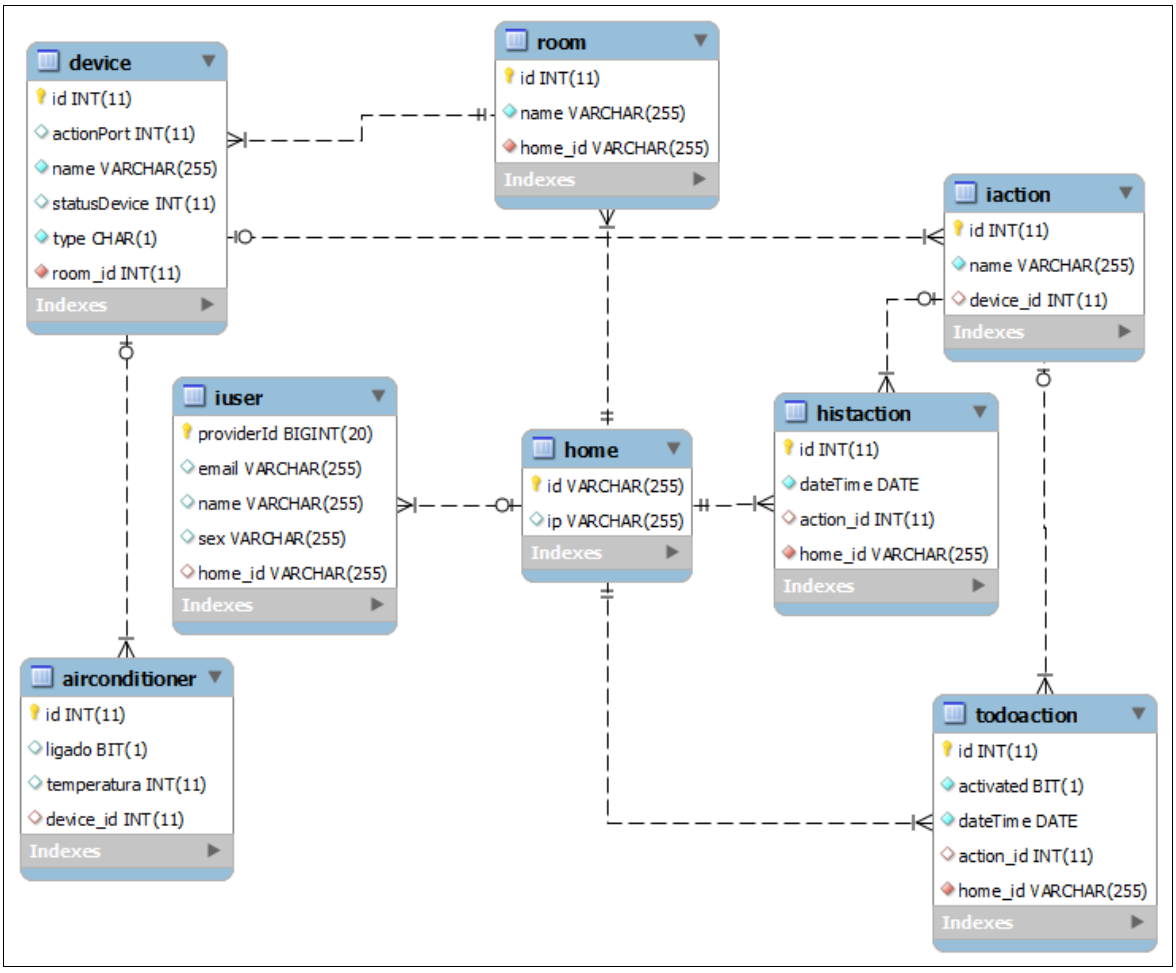


3.2.5 Modelo Entidade Relacionamento

Na Figura 13 é apresentado o modelo de entidade e relacionamento. As tabelas são as utilizadas para gravar a situação da residência, ações do usuário, mineração de padrões e armazenamento de ações automatizadas.

O dicionário de dados está descrito no Apêndice B.

Figura 13 – Modelo entidade relacionamento



A seguir é apresentada uma breve descrição das entidades criadas para o desenvolvimento da aplicação:

a) Home: entidade que armazena as informações de conexão a residência;
b) Room: entidade que armazena o nome dos cômodos de uma determinada residência;
c) Device: entidade que armazena as informações de um determinado equipamento de um cômodo da residência;

d) AirConditioner: entidade que armazena as informações de um condicionador de ar;
e) IUser: entidade que armazena as informações dos usuários da aplicação;
f) IAction: entidade que armazena as informações de uma ação, podendo ser do histórico ou uma ação programada;
g) HistAction: entidade que armazena as informações o histórico de ações;
h) ToDoAction: entidade que armazena as informações as ações programadas para execução automatizada.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A implementação da aplicação foi dividida em sete fases: Persistência, Interface, Autenticação do usuário, Controle da residência, Mineração do histórico de ações, Programação de execução de ações e por fim os testes em *Smartphones* e *Tablet*. Em cada fase foram utilizadas diferentes ferramentas para o desenvolvimento. Para o desenvolvimento da aplicação *web*, foi utilizada a *Integrated Development Environment* (IDE) NetBeans 7.2. A seguir, é detalhado o desenvolvimento bem com as demais ferramentas utilizadas para cada fase da implementação.

3.3.1.1 Persistência

Com o fim de armazenar os dados da aplicação, desde as informações do usuário e da residência, até o histórico de ações em uma determinada residência, foi desenvolvida uma camada de persistência na aplicação. Para desenvolver esta camada foi utilizado o sistema gerenciador de banco de dados MySQL versão 1.1.5, juntamente com a biblioteca Hibernate na versão 3.

O Hibernate é uma biblioteca que permite abstrair o código *Structured Query Language* (SQL) durante o desenvolvimento da aplicação, sendo que este é gerado em tempo de execução da

aplicação, gerando o SQL que serve para um determinado banco de dados, permitindo a troca de dados mais facilmente.

Esta troca de dados é realizada por meio de classes Java que servem de modelos para as tabelas do banco de dados. Através de anotações Java, identifica-se as classes que correspondem a uma tabela no banco de dados, e cada atributo da classe que corresponde a um campo da tabela, sendo também definidos os relacionamentos entre as tabelas. Na Figura 14 é apresentada uma parte do modelo da entidade IUser com estas anotações.

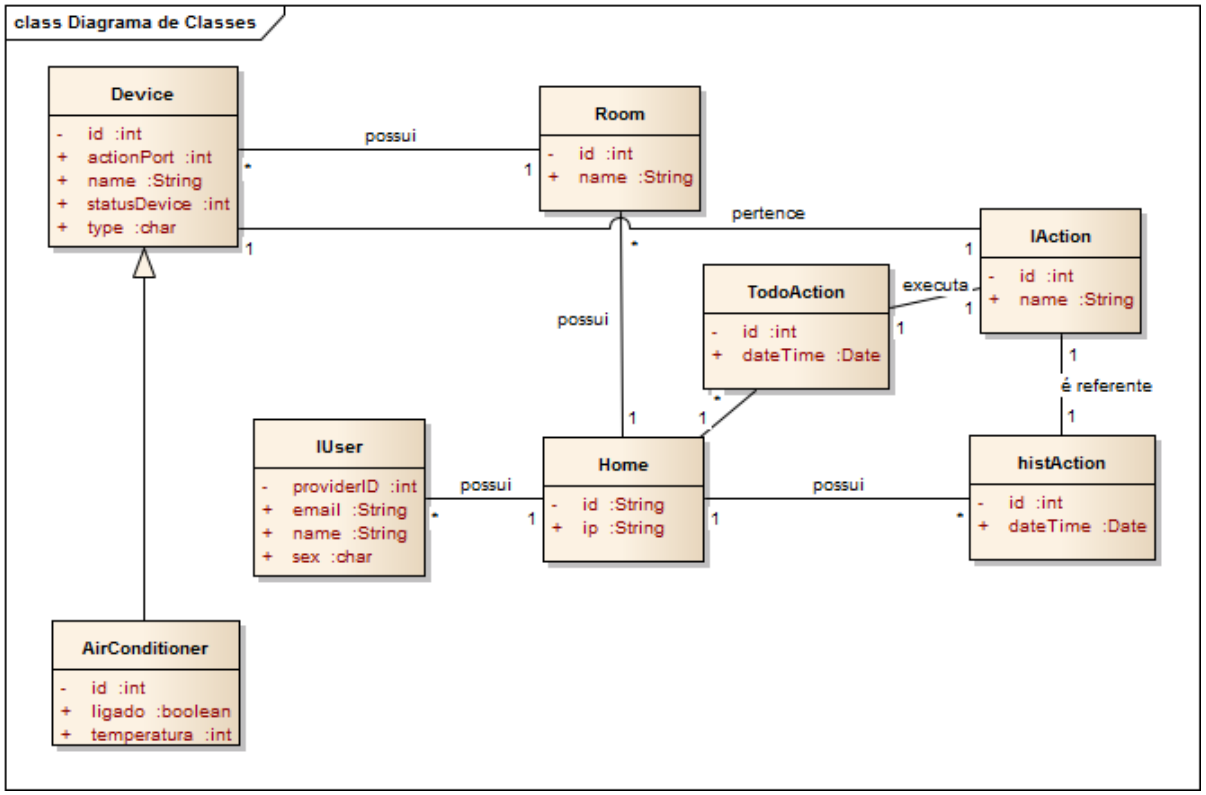
Figura 14 – Classe de modelo de Usuário

```
@Entity
public class IUser implements Serializable {
    @Id
    private long providerId;
    @Column
    private String name;
    @Column
    private String sex;
    @Column
    private String email;
    @ManyToOne
    @JoinColumn(name = "home_id")
    private Home home;


    public String getName() {
        return name;
    }
```

Todos os modelos desenvolvidos estão apresentados no diagrama de classes da Figura 15. Por se tratarem de modelos para a criação de tabelas e para a manipulação destes dados, estas classes não possuem nenhum método além dos métodos de encapsulamento.

Figura 15 – Diagrama de classes da aplicação



A partir deste modelo, foram implementadas as classes que realizam a conexão com o banco de dados, classes estas utilizadas pela camada de interface para realizar a persistência dos dados. Cada modelo criado possui uma classe semelhante a apresentada na Figura 16, na qual estão sendo apresentada a classe referente ao modelo de usuário, com o método que retorna o registro de um usuário com determinado identificador existente na base de dados, e o método que realiza a persistência dos usuários, tanto atualização de um existente, como um novo usuário.

Figura 16 – Classe de persistência de dados

```
public class IUserDao {

    public IUser get(long id) {
        EntityManager em = JpaUtil.get().getEntityManager();
        try {
            return em.find(IUser.class, id);
        } finally {
            em.close();
        }
    }

    public boolean save(IUser u) {
        EntityManager em = JpaUtil.get().getEntityManager();
        EntityTransaction trans = em.getTransaction();
        trans.begin();

        try {
            if (this.get(u.getProviderId()) == null) {
                em.persist(u);
            } else {
                em.merge(u);
            }
            trans.commit();
            return true;
        } catch (Exception ex) {
            trans.rollback();
            return false;
```

Uma vez conectado, o Hibernate realiza a persistência ou extração dos dados do sistema de gerenciamento de banco de dados MySQL.

> Seguindo o padrão SQL-99, o MySQL é o software mais popular quando o assunto é banco de dados livres. Este está sendo o banco de dados mais adquirido do mundo, com 30 mil *downloads* diários e 5 milhões de instalações (entre *websites*, *datawarehouse*, aplicações comerciais, etc). O MySQL é um banco de dados voltado reproduzindo a base de dados que auxilia o modulo de frente de caixa. (COLARES, 2007).

Com a persistência dos dados atende-se o Requisito Funcional 07 e com a utilização do banco de dados MySQL o Requisito Não Funcional 03.

3.3.1.2 Interface

Para o desenvolvimento da interface da aplicação foi utilizado o JavaServer Faces 2.1, que é um *framework* desenvolvido em Java e utilizado para construir interfaces formadas por

componentes voltados para aplicações *web*. Adicionalmente a este, foi utilizada a biblioteca Primefaces 4.0, que é uma biblioteca de código aberto desenvolvida pela PrimeTek na Turquia, que possui vários conjuntos de componentes de interface, que facilitam a criação de interfaces para aplicações que utilizam o JavaServer Faces (PRIMEFACES, 2014).

Os componentes de interface tanto do JavaServer Faces quanto do Primefaces são utilizados simultaneamente, provendo assim um conjunto de funções que facilitam a criação de telas, bem como a interação destas com as rotinas da aplicação. No momento em que estes componentes são carregados na tela, é realizada a verificação de quais cômodos e respectivamente equipamentos (Luz, Janela ou Ar Condicionado) a residência do usuário possui, e apresenta estes na tela do dispositivo.

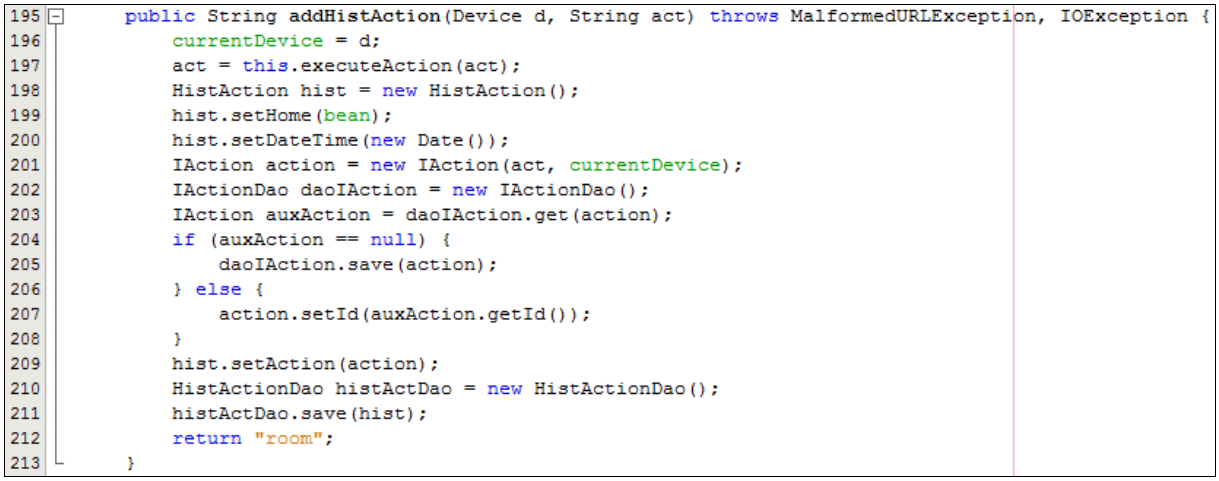
A seguir é apresentada a Figura 17 com um trecho do código da tela de um cômodo, onde na primeira linha é verificado qual o tipo de equipamento que está sendo exibido (no trecho é verificado se é um ar condicionado) e em seguida exibe os respectivos comandos. Com isto atende-se os Requisitos Funcionais do 03 ao 06 e o 09.

Figura 17 – Trecho do código da tela de um cômodo

```
<c:if test="#{device.type == 'a'}">
    <h:form>
        <p:panel header="Ar condicionado" toggleable="true">
            <h:panelGrid columns="2" id="grid" rendered="#{device.airConditioner.ligado == true}">
                <h:outputLabel for="temp" value="Temperatura: #{device.airConditioner.temperatura}" /><h:outputLabel/>
                <p:commandButton  action="#{topHatMB.addHistAction(device, 'temp+')}" value="Temp +" ajax="false"/><br/>
                <p:commandButton  action="#{topHatMB.addHistAction(device, 'temp-')}" value="Temp -" ajax="false"/><br/>
                <p:commandButton  action="#{topHatMB.addHistAction(device, 'swing')}" value="Swing" ajax="false"/><br/>
                <p:commandButton  action="#{topHatMB.addHistAction(device, 'a')}" value="Desligar" ajax="false"/><br/>
            </h:panelGrid>
            <p:outputPanel rendered="#{device.airConditioner.ligado == false}">
                <p:commandButton  action="#{topHatMB.addHistAction(device, 'a')}" value="Ligar" ajax="false"/><br/>
            </p:outputPanel>
        </p:panel>
    </h:form>
</c:if>
```

Cada botão da tela, está vinculado a um método em uma classe denominada *Managed bean* no JavaServer Faces, a qual é acessível pela tela, e executa as ações pressionadas. Esta classe também tem acesso as demais classes da aplicação, como as classes que realizam a persistência. A seguir é apresentada a Figura 18 o método que armazena a ação no histórico da residência. Este método recebe como parâmetro o equipamento no qual a ação será executada, e qual a ação que será executada. A partir da linha 198 até a 209 ele cria um histórico de ação a partir destes parâmetros e então persiste este na base de dados na linha 211.

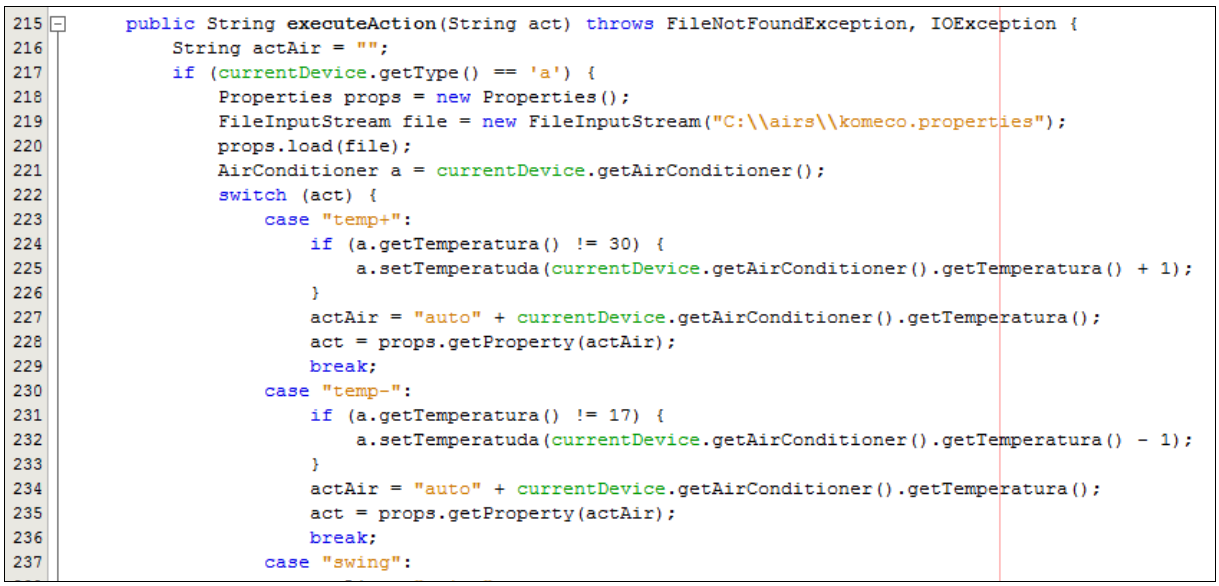
Figura 18 – Código de armazenamento da ação como histórico



```
195     public String addHistAction(Device d, String act) throws MalformedURLException, IOException {
196         currentDevice = d;
197         act = this.executeAction(act);
198         HistAction hist = new HistAction();
199         hist.setHome(bean);
200         hist.setDateTime(new Date());
201         IAction action = new IAction(act, currentDevice);
202         IActionDao daoIAction = new IActionDao();
203         IAction auxAction = daoIAction.get(action);
204         if (auxAction == null) {
205             daoIAction.save(action);
206         } else {
207             action.setId(auxAction.getId());
208         }
209         hist.setAction(action);
210         HistActionDao histActDao = new HistActionDao();
211         histActDao.save(hist);
212         return "room";
213     }
```

Antes de armazenar o histórico, na linha 197, este método chama o método que executa as ações da residência, o qual é apresentado em partes na Figura 19. Este método verifica a ação que está sendo executada, sendo que na parte apresentada, estão sendo verificadas algumas ações do ar condicionado, como aumentar a temperatura na linha 223, diminuir a temperatura na linha 230, ou ativar o modo *swing* na linha 237. Ao fim do método, é montada a URL de chamada da ação em questão, conforme apresentado na seção Controle da residência mais a frente, e realiza a chamada do Arduino. Ambos os métodos estão localizados no principal *Managed bean* da aplicação.

Figura 19 – Trecho do método que executa as ações da residência



```
215     public String executeAction(String act) throws FileNotFoundException, IOException {
216         String actAir = "";
217         if (currentDevice.getType() == 'a') {
218             Properties props = new Properties();
219             FileInputStream file = new FileInputStream("C:\\airs\\komeco.properties");
220             props.load(file);
221             AirConditioner a = currentDevice.getAirConditioner();
222             switch (act) {
223                 case "temp+":
224                     if (a.getTemperatura() != 30) {
225                         a.setTemperatuda(currentDevice.getAirConditioner().getTemperatura() + 1);
226                     }
227                     actAir = "auto" + currentDevice.getAirConditioner().getTemperatura();
228                     act = props.getProperty(actAir);
229                     break;
230                 case "temp-":
231                     if (a.getTemperatura() != 17) {
232                         a.setTemperatuda(currentDevice.getAirConditioner().getTemperatura() - 1);
233                     }
234                     actAir = "auto" + currentDevice.getAirConditioner().getTemperatura();
235                     act = props.getProperty(actAir);
236                     break;
237                 case "swing":
```

3.3.1.3 Autenticação do usuário

A autenticação a aplicação deve, necessariamente, ser segura, garantindo que os dados do usuário não serão acessados indevidamente. Por este motivo, foi optado pela autenticação utilizando a rede social Facebook, pois além da garantia da segurança das informações do usuário (realizada pelo próprio Facebook), tem-se a facilidade de não ser necessário um cadastro de usuário para utilizar a aplicação.

Para realizar esta autenticação, foi utilizada a biblioteca Social Auth 4.6, que auxilia o desenvolvimento da autenticação da aplicação por meio de provedores de autenticação (Facebook, Twitter, Google, entre outros) (SOCIALAUTH, 2014).

Quando utilizada esta biblioteca, cabe a aplicação apenas passar os dados do aplicativo cadastrado no provedor e redirecionar a aplicação para este. Fica a cargo da biblioteca a montagem do *token* necessário para a aplicação, a montagem da *Uniform Resource Locator* (URL) para redirecionar a aplicação. Fica a cargo do provedor exibir a página para informar usuário e senha, autenticar o usuário, e redirecionar a sessão para a URL de sucesso da aplicação. Com isto atende-se os Requisitos Funcionais 01, 02 e o Não Funcional 04.

A seguir na Figura 20 é apresentado uma parte do código para autenticação utilizando a Social Auth, onde na linha 40 são carregadas as informações da aplicação cadastrada no facebook, as quais estão gravadas em um arquivo de propriedades padrão do Social Auth. Na linha 47 define a URL para onde o usuário deve ser redirecionado em caso de sucesso na autenticação e na linha 53 é realizado o redirecionamento para o Facebook.

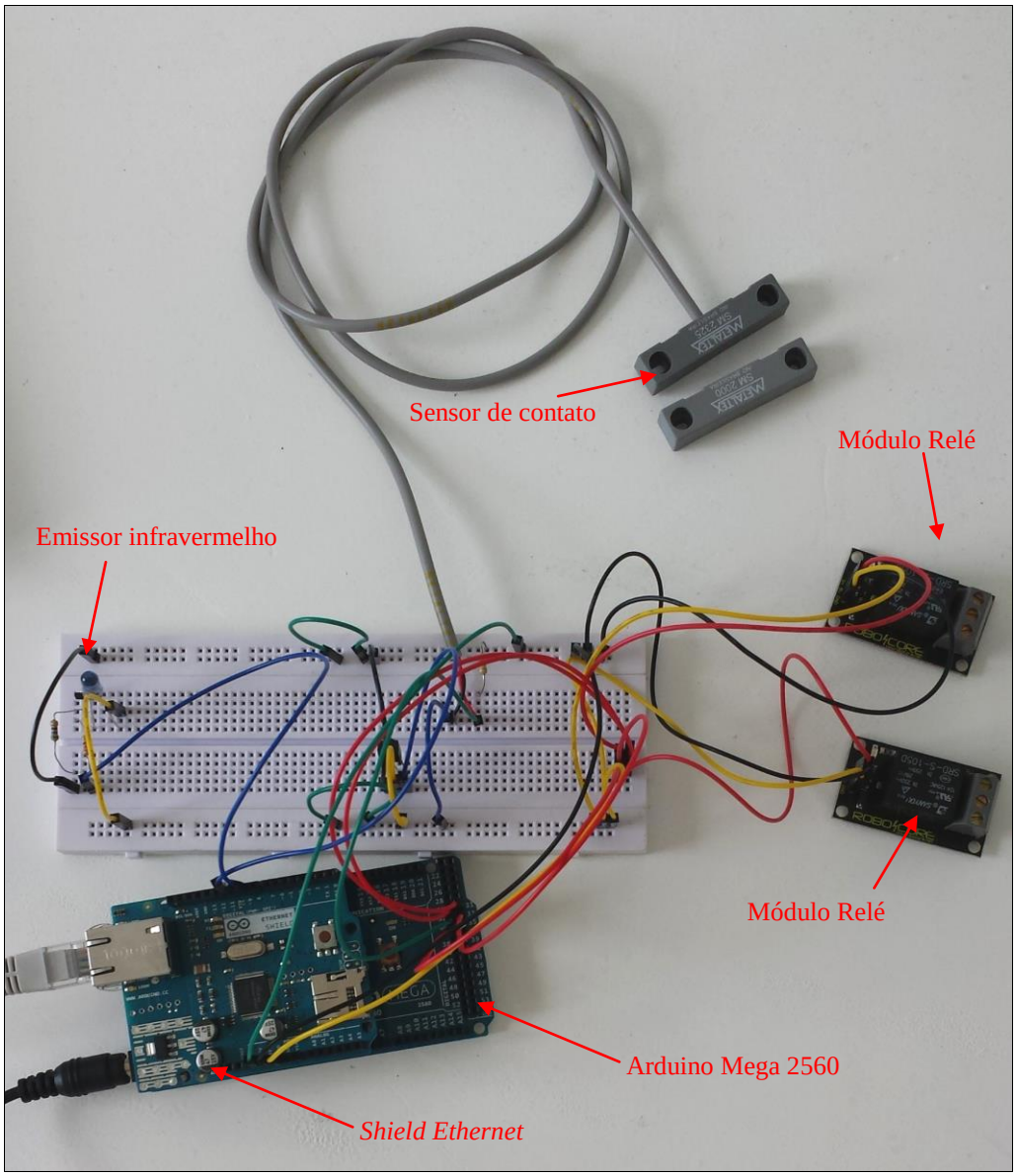
Figura 20 – Código para autenticação utilizando Social Auth

```
37      public void socialConnect(String providerIDSelected) throws Exception {
38          this.setProviderID(providerIDSelected);
39          System.out.println(providerID);
40          SocialAuthConfig config = SocialAuthConfig.getDefault();
41
42          config.load();
43
45          SocialAuthManager manager = new SocialAuthManager();
46          manager.setSocialAuthConfig(config);
47
48          String successUrl = "http://tophat.tcc.br/TopHatApp/home.jsf";
49
50          String url = manager.getAuthenticationUrl(providerID, successUrl);
51
52          HttpSession session = (HttpSession) FacesContext.getCurrentInstance().
53          session.setAttribute("authManager", manager);
54          FacesContext.getCurrentInstance().getExternalContext().redirect(url);
55          session.setAttribute("userSession", this);
      }
```

3.3.1.4 Controle da residência

Para controlar a residência através da aplicação, foi construído um circuito composto por um controlador, o Arduino Mega, um sensor de contato para monitoramento da janela, e três atuadores, sendo um transmissor de infravermelho para controlar condicionador de ar e dois relés para o controle de luzes (Figura 21). Este circuito pode ser ampliado de acordo com a necessidade de cada residência, adicionando sensores e controladores. O desenho e o esquema elétrico deste circuito estão apresentados no Anexo A.

Figura 21 – Foto do circuito utilizado para o desenvolvimento da aplicação



Para controlar o Arduino, foi necessário desenvolver uma classe na linguagem C++, a qual é denominada *sketch*. O desenvolvimento da *sketch* foi realizado na IDE Arduino, distribuída pela produtora da placa. Esta IDE possui poucos recursos, dentre os mais importantes está a conexão com o Arduino via cabo *Universal Serial Bus* (USB) para transferir a *sketch* para o Arduino, bem como conectar o Arduino ao computador permitindo emitir mensagens com o fim de depurar o código (ARDUINO, 2014).

Esta *sketch* possui duas principais divisões, o método *setup* que é executado na inicialização do Arduino, uma única vez, e o método *loop*, que é executado cíclica e ininterruptamente. Além destes dois métodos, podem ser criados novos métodos bem como variáveis.

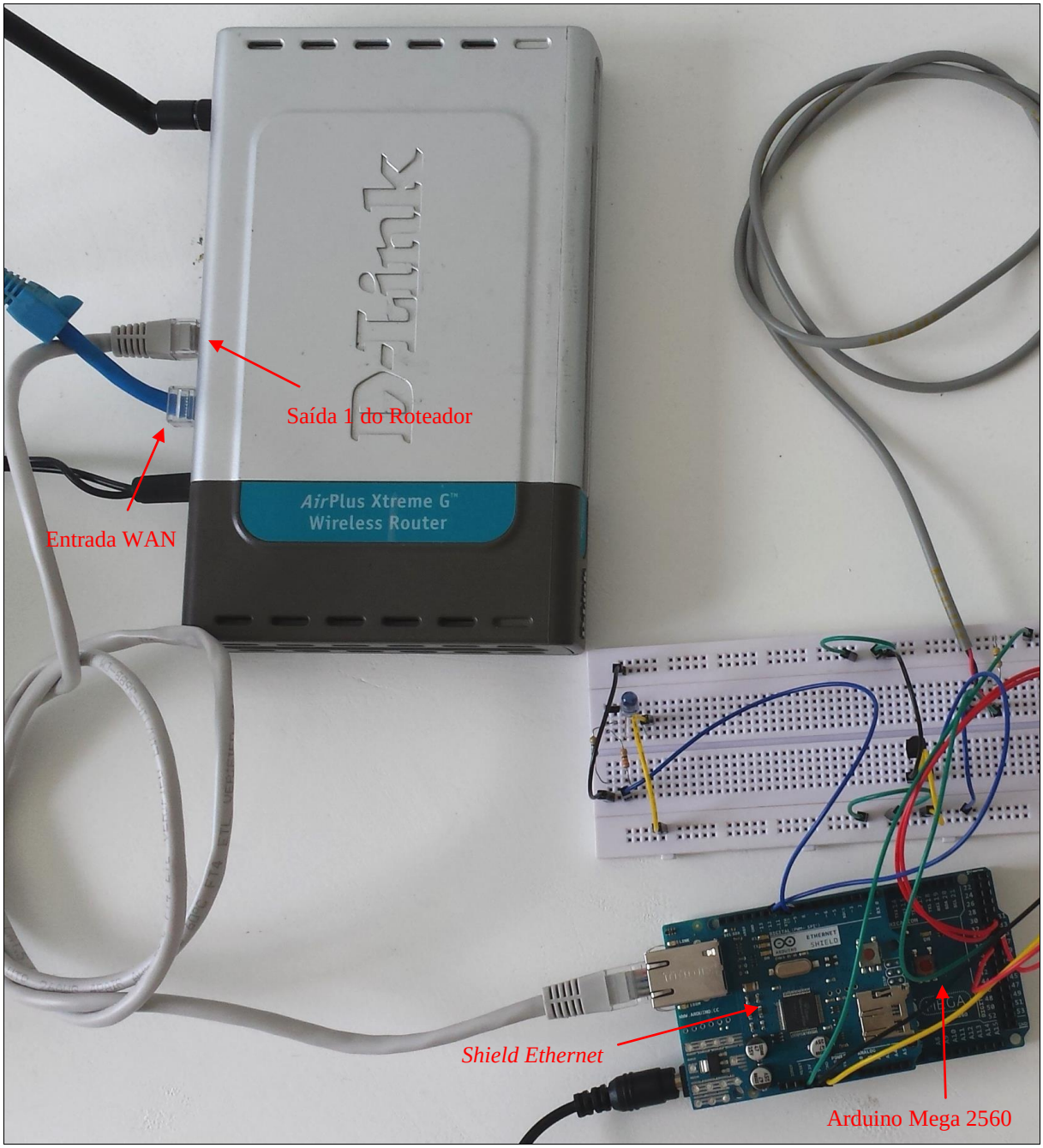
No método *setup* da aplicação, apresentado na Figura 22, define-se qual a forma de operação de cada porta que será utilizada na aplicação, se esta será de saída como as portas das linhas 44 a 51 por exemplo, ou de entrada como as das linhas 53 a 60. Além destas definições, também inicia-se a *shield* Ethernet como um servidor, na linha 69. Para estas definições, utiliza-se de variáveis previamente definidas na *sketch* com os números das portas, código do *Internet Protocol* (IP), entre outros. Para fins de depuração inicia-se também a porta serial na linha 70, que permite enviar mensagens para a IDE a partir da *sketch*.

Figura 22 – Método *setup* da *sketch* do Arduino

```
43  void setup() {
44    pinMode(kL, OUTPUT);
45    pinMode(kW, OUTPUT);
46    pinMode(lL, OUTPUT);
47    pinMode(lW, OUTPUT);
48    pinMode(blL, OUTPUT);
49    pinMode(blW, OUTPUT);
50    pinMode(b2L, OUTPUT);
51    pinMode(b2W, OUTPUT);
52
53    pinMode(kLSt, INPUT);
54    pinMode(kWSt, INPUT);
55    pinMode(lLSt, INPUT);
56    pinMode(lWSt, INPUT);
57    pinMode(blLSt, INPUT);
58    pinMode(blWSt, INPUT);
59    pinMode(b2LSt, INPUT);
60    pinMode(b2WSt, INPUT);
61
62    //Start pin Infra Red
63    pinMode(9, OUTPUT);
64    digitalWrite(9, LOW);
65
66    //start shield Ethernet
67    Ethernet.begin(mac, ip, gateway, subnet);
68    Udp.begin(localPort);
69    server.begin();
70    Serial.begin(9600);
71  }
```

O Arduino é ligado em uma das portas ethernet do roteador da residência como um computador normal, porém, como a porta informada é uma informação fixa, foi necessário configurar o roteador para que, sempre que chamada a porta em questão, fosse apontado para o IP do Arduino no roteador. Será necessário realizar esta configuração a cada residência em que o Arduino for instalado. Segue na Figura 23 a foto do Arduino conectado no roteador a partir da *shield* Ethernet.

Figura 23 – Foto da conexão do Arduino com o Roteador



Logo após o método *setup*, é declarado o método *loop*. Nele foi desenvolvido o código que

realiza o controle da residência. Na parte inicial do método, este passa a verificar ciclicamente o recebimento de uma conexão por parte de um cliente. Esta conexão é realizada por meio do protocolo *Hypertext Transfer Protocol* (HTTP), atendendo assim o Requisito Não Funcional 01. A Figura 24 apresenta o trecho da *sketch*, onde é verificado se há alguma a conexão da com o Arduino.

Figura 24 – Trecho da *sketch* que aguarda a conexão da aplicação

```
73  void loop() {
74    // Create a client connection
75    EthernetClient client = server.available();
76    if (client) {
77      while (client.connected()) {
78        if (client.available()) {
```

O cliente desta conexão é a aplicação *web* desenvolvida, a qual realiza a conexão obedecendo o padrão interpretado pelo Arduino. Este padrão consiste em um ponto de interrogação seguido pelo comando e a porta do dispositivo que deseja-se operar, e no caso dos condicionadores de ar, o respectivo comando, ficando algo semelhante à “?l9”, onde l significa ligar, e 9 a porta do Arduino onde está ligado o dispositivo que deseja-se ligar. Os seguintes comandos estão implementados na *sketch*:

a) “l” para ligar uma luz;
b) “d” para desligar uma luz;
c) “a” para enviar um comando infravermelho para o ar condicionado.

Quando recebida, uma conexão com o padrão acima mencionado obtém-se os comandos passados pela conexão e inicia-se a divisão deste para a execução da ação desejada. Se recebido o comando para ligar uma porta, esta é ligada, se para desligar esta é desligada, e se para enviar um comando para o ar condicionado, o comando é enviado.

A Figura 25 apresenta a parte da *sketch* que identifica se é o comando para ligar e desligar uma luz, nas linhas 95 e 105 respectivamente. Após identificado o comando, é obtida a porta recebida e executa-se o comando, nas linhas 102 e 112.

Figura 25 – Trecho da *sketch* que identifica os comandos ligar e desligar

```
92          if (c == '\n') {
93            for (int i = 0; i < sizeof(search); i++) {
94              if (search[i] == '?') {
95                if (search[i + 1] == 'l') {
96                  port = "";
97                  port += search[i + 2];
98                  if (isDigit(search[i + 3])) {
99                    port += search[i + 3];
100                 }
101                 int portNum = port.toInt();
102                 digitalWrite(portNum, HIGH);
103                 Serial.println("Luz ligada. " + port);
104                 i = sizeof(search);
105               } else if (search[i + 1] == 'd') {
106                 port = "";
107                 port += search[i + 2];
108                 if (isDigit(search[i + 3])) {
109                   port += search[i + 3];
110                 }
111                 int portNum = port.toInt();
112                 digitalWrite(portNum, LOW);
113                 Serial.println("Luz desligada.");
114                 i = sizeof(search);
```

Quando identificado o comando para emitir um sinal infravermelho para o condicionador de ar (comando “a”), deve-se receber juntamente os códigos do comando que deve ser enviado, sendo uma sequência de números separados por vírgula. Para obter esta sequência de números do controle da marca Komeco, foi necessário utilizar uma outra *sketch* específica para este fim, em conjunto com o receptor de infravermelho. Esta *sketch* escreve na tela de depuração da IDE Arduino através da porta serial, o comando recebido pelo receptor infravermelho. Com isto, foi obtido e gravado o comando de cada tecla do controle em um arquivo de propriedades armazenado na aplicação *web*. O circuito para ligação do receptor infravermelho é apresentado junto com o circuito completo no Anexo A.

Além desta *sketch*, foi necessário utilizar um circuito para ampliar o alcance do sinal emitido pelo emissor infravermelho baseado no circuito proposto por Instructables (2014?), porém com a alteração dos resistores. Este circuito é composto por um transistor e dois resistores, sendo apresentado junto ao desenho do circuito no anexo A.

A transmissão do infravermelho através do Arduino, é realizada utilizando a biblioteca IRemote, sendo enviado um sinal no padrão *Raw*. Este padrão consiste no envio da sequência de números do comando desejado. A Figura 26 apresenta o trecho de código que transforma os códigos recebidos em um *array* de números, sendo que da linha 130 a 138 verifica-se caractere por caractere o comando até que seja encontrada uma vírgula, o que então se caracteriza mais um

número do comando infravermelho. Quando verificado todo o comando recebido, este é emitido pelo transmissor infravermelho, na linha 143.

Figura 26 – Trecho da *sketch* de transmissão do infravermelho

```
125                 int buf = port.toInt();
126                 unsigned int raw[buf];
127                 port = "";
128                 char aux;
129                 int comand = 0;
130                 while (aux != '|') {
131                   if (isDigit(aux)) {
132                     port += aux;
133                   } else if (aux == ',') {
134                     raw[comand] = port.toInt();
135                     Serial.print(port);
136                     Serial.print(", ");
137                     port = "";
138                     comand++;
139                   }
140                   aux = search[i];
141                   i++;
142                 }
143                 irsend.sendRaw(raw, buf, 38);
144                 Serial.println("IR emitido.");
145                 i = sizeof(search);
```

Ao fim da *sketch*, após o código de transmissão de infravermelho, foi desenvolvido um código que é executado ciclicamente para enviar os dados da residência para a aplicação via UDP, independente da conexão com o cliente. Estas informações são enviadas no formato de texto único, seguindo um padrão definido para a aplicação. Este padrão consiste na seguinte composição apresentada no Quadro 4.

Quadro 4 – Padrão da mensagem enviada pelo Arduino para a aplicação

|<Identificador> / <Porta> / <Nome Cômodo 1> ; <Tipo dispositivo 1> ; <Porta dispositivo 1> ; <Estado dispositivo 1> ... ; <Tipo dispositivo n> ; <Porta dispositivo n> ; <Estado dispositivo n> / ... <Nome Cômodo n>...|

Cada parte da mensagem apresentada no Quadro 4 deve possuir o conteúdo conforme detalhado a seguir, sendo que cada cômodo pode possuir inúmeros dispositivos e podem existir inúmeros cômodos:

a) identificador: código de identificação da residência;
b) IP: o *Internet Protocol* (IP) *address* da residência;
c) porta: porta TCP a qual o Arduino está conectado;
d) nome cômodo: nome de um cômodo da residência;

e) tipo dispositivo: tipo de um dos dispositivos do cômodo diretamente anterior na mensagem, podendo ser “l” para luzes, “w” para janelas e “a” para condicionadores de ar;

f) porta dispositivo: Porta de execução das ações do dispositivo diretamente anterior na mensagem;

g) estado dispositivo: Estado atual do dispositivo diretamente anterior na mensagem;

As informações para este texto enviado, são obtidas a partir das variáveis declaradas no início do código, e da leitura do estado digital das portas dos dispositivos, sendo um para ligado e zero para desligado. A seguir é apresentado o trecho de código que realiza a montagem e o envio desta mensagem via UDP na Figura 27, sendo que da linha 169 a 177 é montada a mensagem e na linha 184 enviado o pacote.

Figura 27 – Trecho da *sketch* que envia as informações da residência via UDP

```
169       String aux = "|"+homeId + "/" + serverPort
170                    + "/Cozinha;l;" + kL + ";" + digitalRead(kLSt) + ";w;" + kW
171                    + ";" + digitalRead(kWSt) + ";a;" + kA + ";" + 0
172                    + "/Sala de estar;l;" + lL + ";" + digitalRead(lLSt) + ";w;" + lW
173                    + ";" + digitalRead(lWSt) + "/Quarto do casal;l;" + b1L + ";"
174                    + digitalRead(b1LSt) + ";w;" + b1W + ";"
175                    + digitalRead(b1WSt) + "/Quarto do filho;l;" + b2L + ";"
176                    + digitalRead(b2LSt) + ";w;" + b2W + ";"
177                    + digitalRead(b2WSt)+"|";
178       const char *msg = aux.c_str();
179       IPAddress remote(192, 168, 0, 105);
180       // send a reply, to the IP address and port that sent us the packet we received
181       int success = Udp.beginPacket(remote, 1901);
182       //Serial.print("Init Packet: ");
183       //Serial.println(success);
184       success = Udp.write(msg, (int) aux.length());
185       //Serial.print("Pacote enviado: ");
186       //Serial.println(success);
187       success = Udp.endPacket();
```

A mensagem é enviada para o servidor da aplicação, na porta UDP 1901. No servidor da aplicação, esta porta está aberta para receber a mensagem e quando recebida, esta é dividida e cada uma das informações recebidas é persistida no banco de dados nas tabelas da residência em questão. A Figura 28 apresenta um trecho do código da aplicação que realiza o recebimento desta mensagem e a persistência das informações, sendo que da linha 75 a 83 é realizada a preparação do *Datagrama* para o recebimento do pacote, o qual é recebido na linha 88 e o IP do Arduino obtido na linha 89.

Em seguida na linha 90 são verificados o primeiro e o último caractere da mensagem recebida conforme o padrão apresentado no Quadro 4, com o objetivo de garantir que a

mensagem completa tenha chego para a aplicação. A partir de então inicia-se a divisão da mensagem recebida, a criação dos objetos e a persistência dos dados.

Figura 28 – Trecho do código da aplicação que recebe a mensagem UDP do Arduino

```
70      static Thread udp = new Thread() {
71          @Override
72          public void run() {
73              DatagramSocket sock = null;
74              byte[] msg = new byte[256];
75              DatagramPacket pack = new DatagramPacket(msg, msg.length);
76              try {
77                  sock = new DatagramSocket(1901);
78              } catch (SocketException ex) {
79                  org.slf4j.Logger logger = LoggerFactory.getLogger(TopHatMB.class);
80                  logger.error(ex.getMessage(), ex);
81              }
82
83
84              //Recebimento da mensagem
85              while (true) {
86                  try {
87                      sock.receive(pack);
88                      String aux = new String(pack.getData()).trim();
89                      String ip = pack.getAddress().toString();
90                      if (aux.charAt(0) == '|' && aux.charAt(aux.length() - 1) == '|') {
91                          aux = aux.substring(1, aux.length() - 1);
92                      }else{
93                          break;
94                      }
```

Em cada Arduino que for instalado em uma nova residência portanto, deverão ser configuradas as variáveis com as portas dos dispositivos, o endereço IP e a porta TCP do Arduino, e a mensagem UDP passada. A maioria destas informações está concentrada no início da aplicação no bloco de declaração das variáveis, conforme apresentado na Figura 29, sendo que o que cada variável define está descrito no comentário a sua frente.

Figura 29 – Bloco de declaração de variáveis na *sketch*

```
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x9B, 0x36 }; //Endereço mac físico
byte ip[] = { 192, 168, 0, 100 }; // ip na lan
byte gateway[] = { 192, 168, 0, 1 }; // acesso à internet via router
byte subnet[] = { 255, 255, 255, 0 }; //máscara subnet
const int serverPort = 9898;
EthernetServer server(serverPort); //porta do servidor
unsigned int localPort = 8888;

const String homeId = "4_1LFN-C#|"; //Identificador da casa
const int kL = 31; //Cozinha Luz
const int kLSt = 32; //Status Cozinha Luz
const int kW = 33; //Cozinha Janela
const int kWSt = 34; //Status Cozinha Janela
const int kA = 11; //Cozinha Janela
const int lL = 35; //Estar Luz
const int lLSt = 36; //Status Estar Luz
const int lW = 37; //Estar Janela
const int lWSt = 38; //Status Estar Janela
const int b1L = 39; //Quarto1 Luz
const int b1LSt = 40; //Status Quarto 1 Luz
const int b1W = 41; //Quarto1 Janela
const int b1WSt = 42;  //Status Quarto 1 Janela
const int b2L = 43; //Quarto2 Luz
const int b2LSt = 44; //Status Quarto 2 Luz
const int b2W = 45; //Quarto2 Janela
const int b2WSt = 46; //Status Quarto 2 Janela
```

3.3.1.5 Mineração do histórico de ações

Com o objetivo de encontrar padrões nas ações realizadas em uma determinada residência através da aplicação, é realizado diariamente uma mineração dos dados destas ações para encontrar os possíveis padrões existentes. Para realizar esta mineração foi empregada a técnica de mineração Árvore de decisão, detalhada na seção 2.3.

A execução desta técnica consiste em uma série de cláusulas *if* encadeadas, para que se possa tomar a decisão de cadastrar ou não a ação como uma sugestão de automatização. Estes *ifs* verificam entre outras informações se a ação é executada durante o fim de semana ou durante a semana, o período do dia, a hora e o período da hora em que foi executado, podendo ser na primeira meia-hora ou na segunda.

Ao fim destes testes é verificada a quantidade de ocorrências da ação em questão, verificando se esta se trata de um padrão ou não. São consideradas como um padrão as ações que, tomando-se os registros das ações dos últimos três meses, se repetem por cinquenta vezes

quando em dias da semana, e dez quando em fins de semana. Nestes casos, a ação é registrada como uma sugestão de automatização para a residência.

Por serem considerados sempre os três últimos meses no histórico de ações, as propostas de automatização sugeridas se adaptam às mudanças sazonais de comportamento do usuário, como por exemplo, em relação as estações do ano, ou férias. A Figura 30 apresenta um trecho do código de mineração das ações, no qual o primeiro laço separa as ações executadas no fim de semana das executadas em dias da semana, os dois seguintes separam as ações executadas no período da manhã das executadas no período da tarde. Com isto atende-se o Requisito Funcional 08.

Figura 30 – Trecho do código de mineração de ações

```
for (int i = 0; i < data.size(); i++) {
    if (data.get(i).getDateTime().getDay() == 1 || data.get(i).getDateTime().getDay() == 6) {
        weekend.add(data.get(i));
    } else {
        weekdays.add(data.get(i));
    }
}
for (int i = 0; i < weekdays.size(); i++) {
    if (weekdays.get(i).getDateTime().getHours() < 12) {
        weekdaysMorning.add(weekdays.get(i));
    } else {
        weekdaysAfternoon.add(weekdays.get(i));
    }

}
for (int i = 0; i < weekend.size(); i++) {
    if (weekend.get(i).getDateTime().getHours() < 12) {
        weekendMorning.add(weekend.get(i));
    } else {
        weekendAfternoon.add(weekend.get(i));
    }

}
Iterator it = weekendMorning.iterator();
HistAction aux;
```

3.3.1.6 Programação de execução de ações

Uma vez realizada a mineração do histórico das ações realizadas em uma residência conforme detalhado na seção acima, os padrões encontrados, se encontrados, são armazenados como sugestão de automatização para a residência. Uma vez armazenadas estas sugestões, cabe a algum usuário da residência ativar esta sugestão, e quando ativada esta passa a ser executada na recorrência identificada.

Para programar a execução desta ação quando ativada a sugestão, foi utilizada a Quartz Scheduler 2.2.1, que é uma biblioteca Java que vem sendo desenvolvida a mais de uma década,

por uma comunidade de software livre. Esta biblioteca permite programar a execução de rotinas, criando um calendário de execuções, as quais são chamadas de *trigger*, facilitando o gerenciamento de execuções programadas. Vinculados a estas *triggers* fica um *job* com o objeto da ação que deve ser executada no momento em questão (QUARTZ SCHEDULER, 2014).

Para a configuração da periodicidade da execução destas *triggers*, é utilizada uma combinação de caracteres que definem quando esta deve ser executada, semanalmente, mensalmente, sempre em um determinado dia do mês, entre outros. Na Figura 31 é apresentado o trecho de código que define a periodicidade da tarefa, mais especificamente na linha 301, utilizando-se da data da ação para obter a hora e o minuto em que esta deve executar. Ao fim, define se deve ser executado nos dias da semana ou apenas nos fins de semana, com a adição dos textos "2-6" na linha 307 ou "SUN-SAT" na linha 305.

Figura 31 – Código que define a periodicidade da execução das tarefas

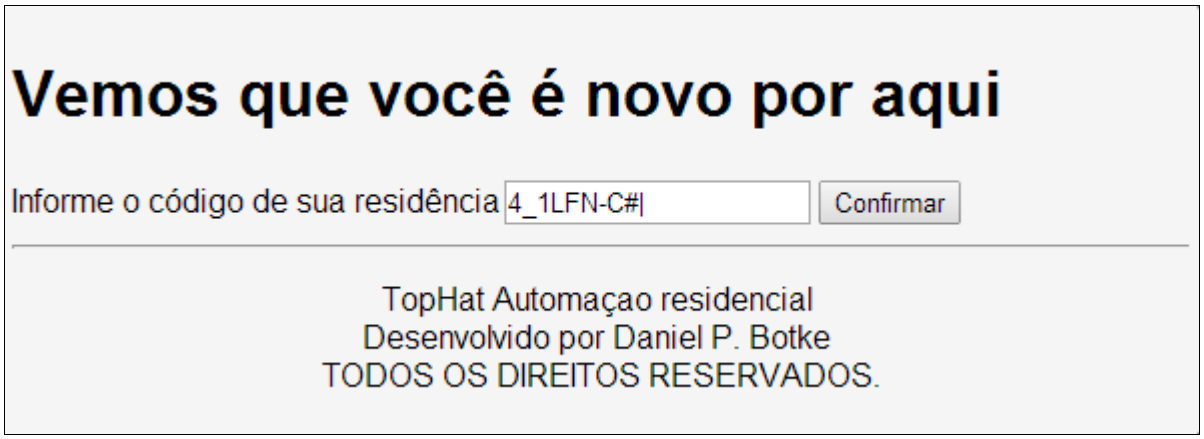
```
301                 String str = "0 " + toDoAction.getDateTime().getMinutes() +
302                         " " + toDoAction.getDateTime().getHours() + " ? * ";
303
304                 if (toDoAction.getDateTime().getDate() == 1) { //Fim de semana
305                     str += "SUN-SAT";
306                 } else if (toDoAction.getDateTime().getDate() == 2) {//Dias de semana
307                     str += "2-6";
308                 }
309                 CronTrigger trigger;
310                 trigger = TriggerBuilder.newTrigger()
311                         .withIdentity(""+toDoAction.getId(), "group1")
312                         .withSchedule(CronScheduleBuilder.cronSchedule(str))
313                         .build();
314
```

3.3.2 Operacionalidade da implementação

Para o uso da aplicação, o usuário necessitará inicialmente realizar sua autenticação à aplicação utilizando uma conta no Facebook. Caso seja o primeiro acesso à aplicação utilizando a conta em questão, o Facebook apresentará uma tela questionando se o usuário permite que a aplicação acesse as informações de seu perfil, devendo confirmar a opção para acessar a aplicação.

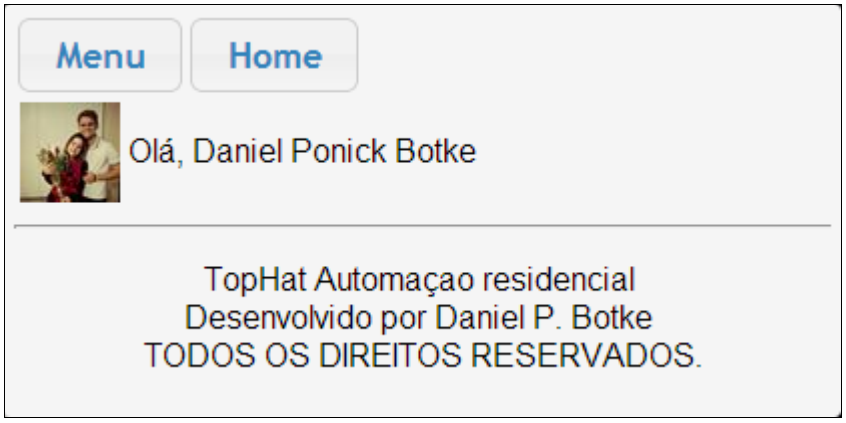
Neste momento, é verificado se a conta utilizada já corresponde a um usuário da aplicação, caso não corresponda, é apresentada uma tela solicitando que seja informado o identificador da residência pertencente ao usuário, conforme Figura 32.

Figura 32 – Tela de um novo usuário



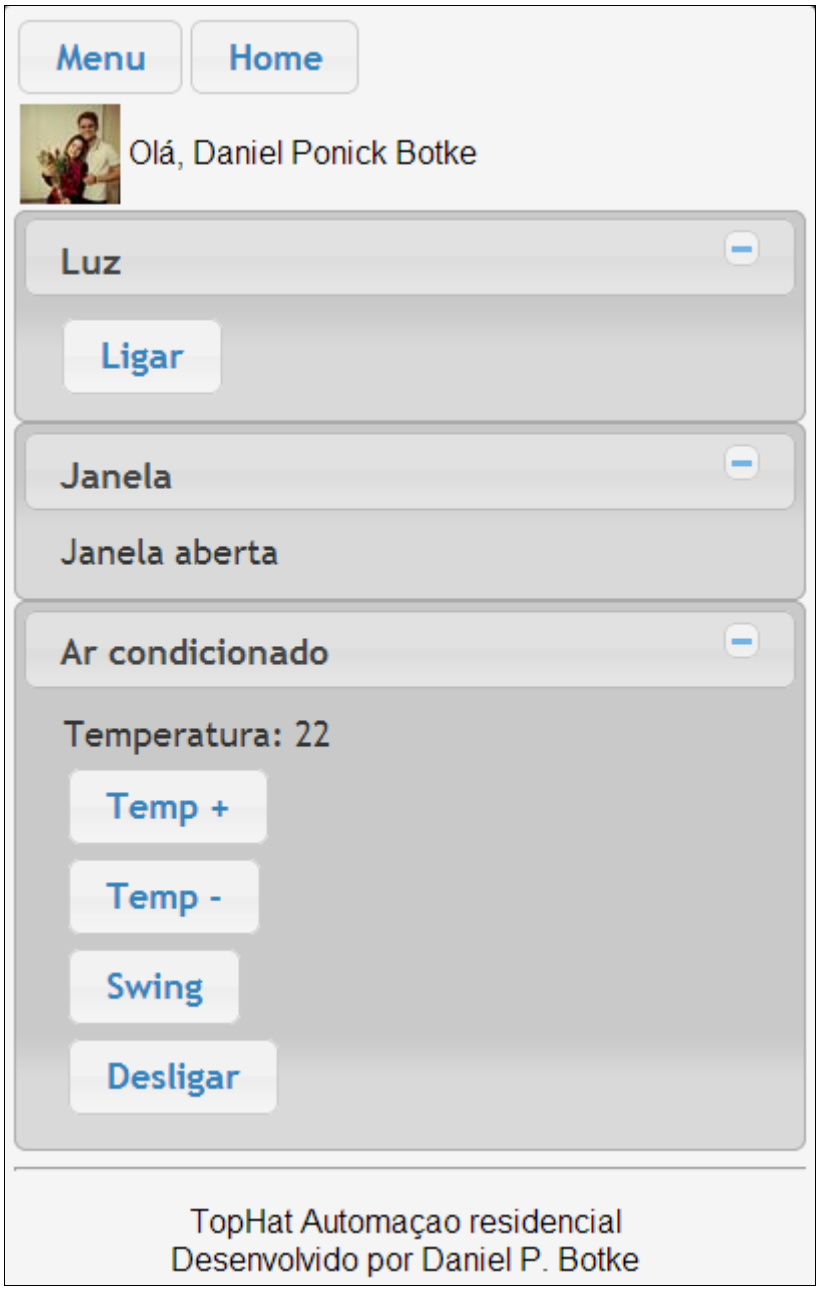
Caso o identificador informado corresponda a alguma residência cadastrada na aplicação, é apresentada a tela inicial da residência, caso contrário, é solicitado ao usuário que este informe um código válido. Na tela inicial da residência, é apresentado um menu com todos os cômodos existentes na residência, a opção de acesso às ações programáveis e a opção de sair da aplicação, conforme Figura 33.

Figura 33 – Tela inicial da residência



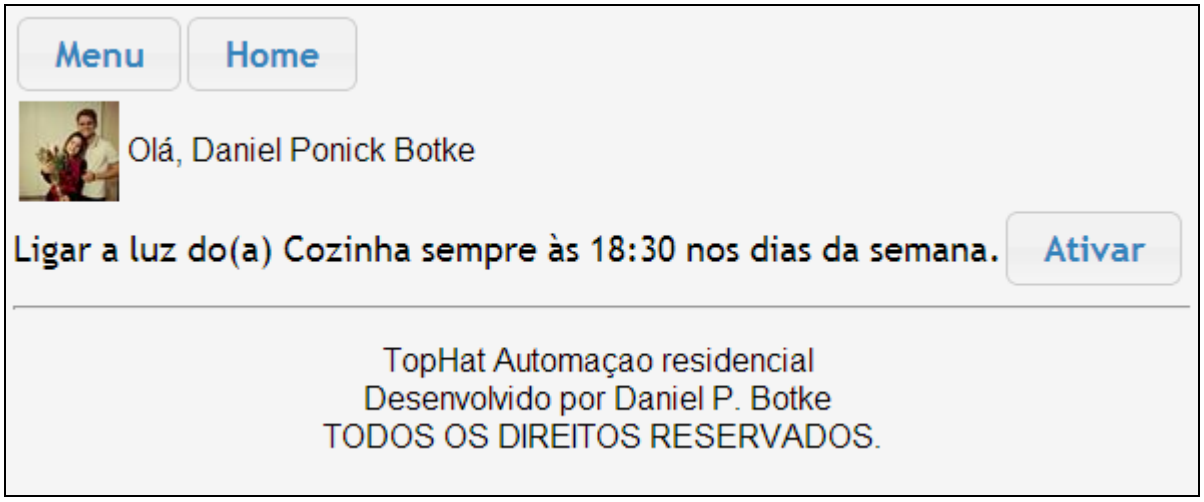
Ao acessar um dos cômodos existentes na aplicação, são apresentados todos os equipamentos existentes neste cômodo, com sua situação atual e respectivos comandos, conforme Figura 34. Ao pressionar algum comando nesta tela, este é enviado ao Arduino que então o executa na residência.

Figura 34 – Tela de um cômodo da casa



Conforme já explanado anteriormente, quando executada alguma ação na aplicação, esta é armazenada. Diariamente executa-se uma rotina para verificar a existência de padrões nestas ações, para então sugerir automatizações dos padrões encontrados. Esta sugestão ao usuário é realizada através da tela de ações programadas, onde são apresentados todos os padrões encontrados nas ações da residência, com a opção de ativá-los ou desativá-los, conforme Figura 35.

Figura 35 – Tela de ações programadas



Por fim, ao pressionar a opção Sair, retorna-se a tela inicial da aplicação, sendo necessário realizar a autenticação novamente para acessar a aplicação.

3.4 RESULTADOS E DISCUSSÃO

O desenvolvimento desta aplicação permitiu a automatização e monitoramento de alguns dispositivos de uma residência, sendo as luzes, condicionadores de ar e janelas. Como o Arduino envia suas informações repetidamente para a aplicação, não há necessidade de um IP fixo para esta automação, sendo que o Arduino envia seu código IP junto destas informações. A única configuração necessária é a do roteador, para que este direcione para o IP interno do Arduino sempre que chamada a porta onde este está conectado, porta esta que também é passada pelo Arduino junto com as demais informações.

Por se tratar de um hardware robusto, o Arduino pode permanecer por um longo tempo ligado sem que seja necessário reiniciá-lo. A placa consegue gerenciar os acionadores, bem como emitir o sinal de infravermelho para o ar condicionado.

Como a interface e o processamento da aplicação estão desvinculados do Arduino, e a aplicação é instalada e executada em um servidor único, sua atualização é facilitada quando necessária. O código gravado no Arduino destina-se somente a retornar as informações da residência, de cada cômodo e equipamento para a aplicação, e para executar as ações solicitadas através da aplicação.

Visto que esta aplicação foi desenvolvida na plataforma *web*, tem-se com isso todas as vantagens e recursos já oferecidos pela plataforma, como a diversidade de plataformas de acesso, centralização da manutenção da aplicação e assim por diante. Outro resultado obtido foi a construção de uma aplicação adaptável, ou seja, ela é válida para qualquer configuração de qualquer residência, independentemente da quantidade de cômodos ou equipamentos.

Para este envio das informações do Arduino periodicamente, é utilizado o protocolo UDP, o qual não é orientado a conexão. Com isto não se tem o problema da concorrência de várias residências tentarem atualizar suas informações ao mesmo tempo, esperando uma conexão, pois a partir do momento que a aplicação estiver livre, esta receberá a mensagem a atualizará a residência.

Na parte da mineração de dados, por serem considerados sempre os três últimos meses no histórico de ações, as propostas de automatização sugeridas se adaptam às mudanças sazonais de comportamento do usuário, como por exemplo, em relação as estações do ano, ou férias.

Em relação a segurança das informações do usuário, e a obtenção destas, isto é realizado pelo provedor de autenticação, que nesta aplicação foi utilizado o Facebook. Isto faz com que a aplicação não tenha de se preocupar com a segurança destas informações, nem ter uma tela de cadastro específica para o usuário, simplificando a aplicação.

Também em relação a segurança, mas segurança da aplicação, no caso de alguma pessoa mal intencionada querer obter o controle da residência, além da segurança proporcionada pela autenticação do usuário, tem-se a vantagem de que a chamada para o Arduino, é realizada pela aplicação que está no servidor, dificultando assim o descobrimento do padrão utilizado para a conexão com o Arduino, bem como seu IP e porta.

3.4.1 Comparativo com trabalhos correlatos

A seguir é apresentado um comparativo entre os resultados obtidos neste trabalho e os seus trabalhos correlatos. A fim de reduzir o foco da comparação, serão considerados os dois trabalhos correlatos mais recentes, sendo o de Reiter Júnior (2006) e o de Gadotti (2010).

O comparativo é apresentado no Quadro 5.

Quadro 5 – Comparativo entre o trabalho atual e correlatos

	Trabalho Atual	**Reiter (2006)**	**Gadotti (2010)**
Linguagem de programação	Aplicação: Java; Placa controladora: C++.	Aplicação: Delphi 7; Placa controladora: C.	Aplicação: C *Sharp*; Placa controladora: C.
Plataforma	*Web*	*Desktop*	Twitter
Controles disponíveis	• Controle de acionadores de equipamentos eletrônicos (luzes); • Ar condicionado; • Monitoramento de janelas.	Controle de acionadores de equipamentos eletrônicos (tomadas e luzes);	Controle de acionadores de equipamentos eletrônicos (tomadas e luzes).
Placa controladora	Arduino Mega + *shield Ethernet*	MCS51	FEZ + *shield Ethernet*
Vantagens	• Pode ser acessado de qualquer dispositivo que possua um navegador *web* homologado; • Sugere ações automatizadas na residência com base na análise do histórico de ações na residência; • Fácil configuração e operação, sendo necessário apenas configurar a residência no Arduino através de uma *String* única.		• Não necessita de uma interface para ser operado, uma vez que é operado através do Twitter; • Não necessita obrigatoriamente de conexão à internet uma vez que é possível realizar *tweets* via SMS; • Várias formas de acesso disponibilizadas pelo próprio Twitter.
Desvantagens	Necessita de conexão com a internet.	• Necessidade de montar e instalar um hardware para cada equipamento eletrônico que	• Vulnerabilidade quanto a possíveis impactos de uma alteração na biblioteca

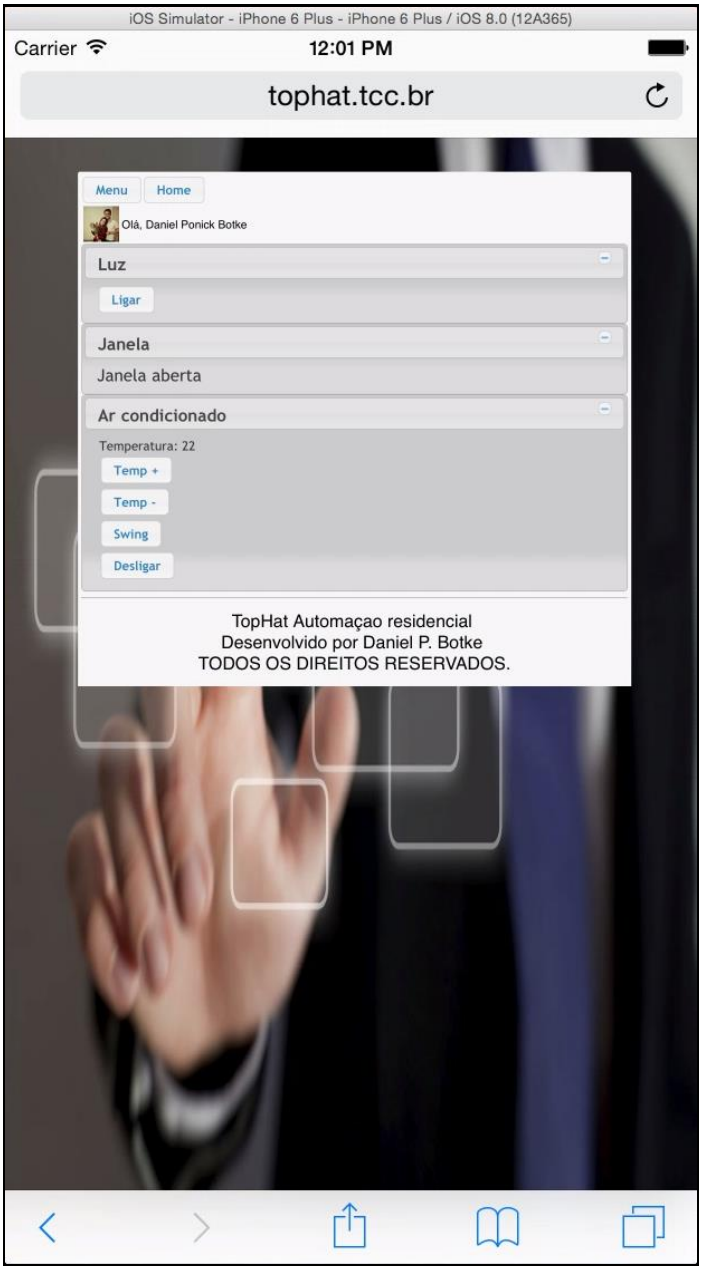
		deseja-se controlar; • Não é possível acessar a aplicação de fora da residência; • Difícil configuração da aplicação, por ser necessário criar manualmente uma série de arquivos de texto com as informações da aplicação e usuário.	do Twitter; • Necessita-se cadastrar muitos arquivos manualmente para o funcionamento do sistema. • O usuário precisa decorar os comandos para escrevê-los no Twitter.

3.4.2 Testes em *Smartphones*

A fim de garantir o funcionamento da aplicação nos sistemas operacionais IOS e Android, conforme Requisito Não Funcional 02, foram realizados testes da aplicação nestas duas plataformas. Para realizar estes testes foi utilizado o emulador de navegadores de dispositivos móveis BrowserStack, uma ferramenta *web* que permite emular os principais dispositivos móveis do mercado. Esta ferramenta é utilizada por diversas empresas importantes e conhecidas do meio tecnológico para realizar os seus testes de aplicações *web*, sendo uma ferramenta de alta confiança e que reproduz com fidelidade as características dos dispositivos móveis (BROWSERSTACK, 2014?).

Foram realizados testes na ferramenta nas versões 8, 7 e 6 do IOS e nas versões 4.4, 4.3 e 4.2 do Android, sempre utilizando o navegador padrão de cada plataforma. Foram emulados os Iphones 6 Plus, 5s e 5, o IPad Air, os Samsung Galaxy S4 e S5 e o Nexus 4. Todos os testes foram executados com sucesso, e a aplicação mostrou-se completamente funcional nestas plataformas, atendendo portanto o Requisito Não Funcional 02. Na Figura 36 é apresentada uma imagem da tela de um cômodo nos testes realizados no IOS 8.

Figura 36 – Teste da aplicação no IOS



Na Figura 37 é apresentada a mesma tela nos testes realizados no Android 4.4.

Figura 37 – Teste da aplicação no Android

3.4.3 Dificuldades encontradas

Durante o desenvolvimento da aplicação foram encontradas algumas dificuldades que impediam de alguma forma a continuidade do desenvolvimento, da aplicação ou do seu funcionamento pleno. Estas dificuldades tiveram portanto, de serem superadas para que a aplicação pudesse ser finalizada e seu funcionamento fosse o esperado em sua totalidade. A seguir são apresentadas algumas das dificuldades que foram consideradas mais cruciais para o

desenvolvimento da aplicação.

A primeira dificuldade encontrada foi durante no desenvolvimento da autenticação da aplicação via Facebook, a qual não estava funcionando. Descobriu-se então que para executar uma aplicação localmente e realizar a autenticação via Facebook, era necessário alterar o nome do *host* local, e informar este como domínio da aplicação no cadastro do Facebook, utilizando sempre a porta 80 no servidor de aplicações. Além disso, se tentava acessar as informações do usuário dentro do método de conexão com o Facebook, o que não é possível.

Outra dificuldade encontrada foi com o recebimento dos pacotes UDP enviados pelo Arduino pela aplicação. Os pacotes estavam sendo enviados e recebidos pelo computador corretamente, no entanto a aplicação não "percebia" a chegada destes pacotes. Após pesquisas e investigações com outros sistemas de recebimento de pacote UDP, foi identificado que para receber estes pacotes, era necessário a criação de uma *Thread* Java na aplicação exclusiva para este recebimento.

Por fim, quando a aplicação estava emitindo o comando infravermelho para controlar o ar condicionado, os comandos não estavam sendo executados pelo equipamento. Com o intuito de verificar se o alcance do sinal do infravermelho pudesse estar curto para alcançar o ar condicionado, foram realizados testes com um repetidor de sinal e com diferentes circuitos para amplificar o sinal, no entanto o ar condicionado ainda assim não executava as ações enviadas.

Após mais algumas pesquisas e testes, foi identificado que o *led* receptor de infravermelho que foi utilizado para obter os comandos do controle não era o correto, sendo que estava sendo utilizado o *led* simples de dois pinos, que não modulariza o sinal recebido. Foi então realizado um teste com o *led* que modulariza o sinal recebido antes de enviá-lo para a aplicação, fazendo com que o pacote correto fosse recebido, permitindo o controle do ar condicionado corretamente.

Além destas, diversas outras dificuldades de programação tiveram de ser superadas, principalmente na *sketch* do Arduino, a qual teve de ser desenvolvida em C++, linguagem de programação desconhecida pelo autor até então. Mas com a superação destas dificuldades, foi possível desenvolver e executar a aplicação por completo conforme especificado.

4 CONCLUSÕES

Neste trabalho é apresentada uma aplicação que permite o controle de alguns equipamentos de uma residência a partir de uma aplicação *web*. A aplicação permite que sejam controladas luzes, condicionadores de ar e monitoradas as janelas.

A situação atual destes equipamentos pode ser consultada a qualquer momento da aplicação. Isto permite que o usuário tenha sempre uma visão atualizada da sua residência, se foi esquecido algum equipamento ligado ou aberto, ou se existe algum movimento suspeito na residência.

Outra funcionalidade disponibilizada é a automatização de ações que são executadas repetidas vezes em um mesmo horário e dia, caracterizando-se uma rotina. Esta automatização é realizada com base na análise do histórico de ações da residência, e a partir de então, cabe ao usuário ativar ou não os padrões identificados. Esta automatização facilita o dia a dia de uma residência, pois além de não precisar estar na residência ou se movimentar para executar alguma das ações disponibilizadas, quando esta é automatizada, nem mesmo é necessário acessar a aplicação para que esta ocorra.

Outra vantagem desta aplicação, é o fato desta ser *web*, não sendo necessário estar na residência para controlá-la ou monitorá-la. Além disso, o acesso à aplicação pode ser feito por meio qualquer dispositivo que possua um navegador *web* homologado.

Por se tratar de uma aplicação *web*, foi utilizado o banco de dados MySQL, bastante utilizado para este tipo de aplicação. A ferramenta mostrou-se muito eficiente para o desenvolvimento e uso da aplicação, com uma fácil interação e rápida atualização dos dados.

Além do banco de dados, as bibliotecas e *framework* utilizados mostraram-se eficientes e atenderam a necessidade da aplicação, sendo que cada uma se destacou por algum motivo. O JavaServer Faces e o Primefaces, permitiram que fosse criada uma interface *web* baseada em componentes, proporcionando uma facilidade de atualização e manutenção. A Social Auth permitiu de forma prática garantir a segurança das informações do usuário por meio do uso de provedores de autenticação. A Quartz com suas diversas possibilidades de combinação de programação permitiu que fossem facilmente programadas as ações rotineiras de uma residência. E por fim o Hibernate, proporcionou facilidade na integração entre a aplicação e o banco de dados.

A opção pela placa controladora Arduino demonstrou-se acertada, por se tratar de uma plataforma que está a bastante tempo no mercado, e com uma grande comunidade de usuários,

se tornando assim uma plataforma robusta. Além disso, foi considerado também a facilidade de expansão dos controles oferecidos pela aplicação e o seu baixo custo.

Para o desenvolvimento deste trabalho, foi adquirido um Arduino Mega 2560, uma *shield* Ethernet, um conjunto de peças eletrônicas como fios e resistores, dois módulos relés, um sensor de contato, e dois sensores infravermelho, totalizando aproximadamente R$ 560,00.

Os controles oferecidos por este trabalho podem além de possibilitar melhorias na execução de atividades cotidianas em uma residência, incentivar pesquisas e outras implementações utilizando-se de microcontroladores, visto que a automação é uma realidade dos nossos dias, e as organizações, e usuários finais tem cada vez mais usufruído delas. Outras pesquisas acadêmicas que podem ser realizadas são em relação ao impacto social com o uso deste tipo de aplicação.

Diante dos resultados obtidos no desenvolvimento deste trabalho, avalia-se positivamente a aplicação de automação de residências, demonstrando que seu uso facilitará a execução das atividades do cotidiano de uma residência, cumprindo assim, os objetivos abordados neste trabalho.

4.1 EXTENSÕES

Para trabalhos futuros, surge a oportunidade de complementar a solução, agregando novas funções à aplicação. Como sugestão pode-se implementar o suporte para outros equipamentos de uma residência como um televisor, uma cortina elétrica, tomadas, entre tantos outros. Como o Arduino faz o controle de acionadores, sendo estes os relés, pode-se facilmente utilizar esta mesma abordagem para o controle de tomadas.

Outra sugestão seria implementar outras opções de autenticação utilizando os provedores homologados pela Social Auth, ampliando as opções do usuário. Em conjunto poderia ser implementado a definição de um usuário administrador da residência, e que este tenha acesso a informações diferenciadas, como tempo que cada equipamento fica ligado, quem efetua as ações, entre outros.

Seria interessante também ampliar o código de mineração de dados, encontrando padrões mais específicos, possibilitando uma maior veracidade nas ações sugeridas. Em conjunto poderia ser implementado a possibilidade de editar uma ação sugeridas.

REFERÊNCIAS

ARDUINO. **ARDUINO**: Products. [Turim], 2014. Disponível em: <http://arduino.cc/en/Main/Products>. Acesso em: 03 set. 2014.

AXELSON, Jan. **The microcontroller idea book**: circuits, programs & applications featuring the 8052-BASIC microcontroller. Madison: Lakeview Research, c1997.

BOLZANI, Caio Augustus Morais. **Residências inteligentes**. São Paulo: Livraria de Física, 2004.

BESEN, Nelson. **Sistema Domótico para Automação e Controle de um Cômodo Residencial**. 1996. 69 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais. Universidade Regional de Blumenau, Blumenau.

BROWSERSTACK. **Live, Web-Based Browser Testing**. [S.l], [2014?]. Disponível em: <http://www.browserstack.com/>. Acesso em: 18 nov. 2014.

CAMILO, Cássio Oliveira; SILVA, João Carlos da. **Mineração de Dados**: Conceitos, Tarefas, Métodos e Ferramentas. [Goiânia], 2009. Disponível em: <http://www.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_001-09.pdf>. Acesso em: 03 set. 2014.

CENSI, Ângela. **Sistema para Automação e Controle Residencial Via E-mail**. 2001. 58 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais. Universidade Regional de Blumenau, Blumenau.

COLARES, Flávio Martins. **Análise comparativa de banco de dados gratuitos**. 2007. 74 f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) - Faculdade Lourenço Filho, 2007.

DIAS, César Luiz de Azevedo; PIZZOLATO, Nélio Domingues. **DOMÓTICA**: Aplicabilidade e Sistemas de Automação Residencial. Rio de Janeiro, 2004. Disponível em: < http://www.essentiaeditora.iff.edu.br/index.php/vertices/article/viewFile/98/86>. Acesso em: 03 set. 2014.

FORESTI, Henrique Braga. **Microcontroladores**. [S.l.], 2013. Disponível em: <http://robolivre.org/conteudo/microcontroladores>. Acesso em: 03 set. 2014.

GADOTTI, Eduardo F. **Sistema para Automação e Controle Residencial Via Twitter**. 2010. 56 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Exatas e Naturais. Universidade Regional de Blumenau, Blumenau.

INSTRUCTABLES. **Puzzlebox Orbit**: Using an Arduino to Control an Infrared Helicopter. [S.l.], [2014?]. Disponível em: < http://www.instructables.com/id/Puzzlebox-Orbit-Using-an-Arduino-to-Control-an-In/step6/Building-the-Circuit/ >. Acesso em: 15 nov. 2014.

MARTE, Claudio Luiz. **Automação Predial**: A Inteligência Distribuída Nas Edificações. São Paulo: Carthago & Forte, 1995.

MECATRÔNICA ATUAL. **Atuadores Pneumáticos**. São Paulo, 2013. Disponível em: <http://www.mecatronicaatual.com.br/educacao/1070-atuadores-pneumticos?showall=&limitstart=0>. Acesso em: 03 set. 2014.

MONK, Simon. **Programação com Arduino**: começando com sketches. Porto Alegre: Bookman, 2013.

POZZER, Cesar Tadeu. **Aprendizado por Árvores de Decisão**. Santa Maria, 2006. Disponível em: < http://www-usr.inf.ufsm.br/~pozzer/disciplinas/pj3d_decisionTrees.pdf >. Acesso em: 24 out. 2014.

PRIMEFACES. **Why Primefaces**. [Turquia], 2014. Disponível em: < http://www.primefaces.org/whyprimefaces >. Acesso em: 10 nov. 2014.

QUARTZ SCHEDULER, **What is Quartz Scheduler?**. [S.l.], 2014. Disponível em: < http://quartz-scheduler.org/generated/2.2.1/html/qs-all/#page/Quartz_Scheduler_Documentation_Set%2Fco-abt_what_is_quartz_scheduler.html%23wwconnect_header >. Acesso em: 10 nov. 2014.

REITER JÚNIOR, René A. **Sistema de automação residencial com central de controle microcontrolada e independente de PC**. 2006. 91 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de ciências exatas e naturais. Universidade Regional de Blumenau, Blumenau.

REZENDE, Solange Oliveira. **Mineração de dados**. São Leopoldo, 2005. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/enia/2005/0102.pdf>. Acesso em: 03 set. 2014.

SCHNITZ, Hubert; CARVALHO, Ruy de Quadros. **Automação, competitividade e trabalho**: a experiência internacional. São Paulo: Hucitec, 1988.

SOCIALAUTH, **Socialauth**. [S.l.], 2014. Disponível em: < http://code.google.com/p/socialauth/ >. Acesso em: 10 nov. 2014.

APÊNDICE A – Descrição dos Casos de Uso

Este apêndice do trabalho apresenta a descrição dos casos de uso conforme previstos nos diagramas apresentados na seção 3.2.2, conforme Quadro 6.

Quadro 6 – Descrição dos casos de uso

UC03 - Controlar luzes
Permite ao usuário controlar as luzes conectadas à aplicação, bem como visualizar o estado em que estas se encontram no momento. O usuário poderá em uma única tela visualizar todas as luzes de um determinado cômodo.

Constraints
Pré-condição. O usuário esteja autenticado à aplicação.
Pré-condição. As luzes estão conectadas ao Arduino e configuradas por cômodos.
Pós-condição. A luz deve mudar seu estado de acordo com o escolhido.
Pós-condição. O estado da luz deve ser atualizado na aplicação.

Cenários
Acessar luz {Principal}.
1. Sistema apresenta na tela todos os cômodos da casa.
2. O usuário seleciona o cômodo desejado para a operação.
3. A aplicação apresenta todas as luzes configuradas para o cômodo.
4. O usuário seleciona a luz desejada para a operação.
5. A aplicação apresenta o estado atual da luz e a operação correspondente ao estado inverso.

Acender a luz {Alternativo}.
No passo 5, se a luz está apagada.
5.1. O usuário pressiona o botão para acender a luz.
5.2. A aplicação acende a luz pressionada.
5.3. A aplicação grava a operação na base de dados histórica.

Apagar a luz {Alternativo}.
No passo 5, se a luz está acesa.
5.1. O usuário pressiona o botão para apagar a luz.
5.2. A aplicação apaga a luz pressionada.
5.3. A aplicação grava a operação na base de dados histórica.

UC04 - Controlar condicionadores de ar
Permite ao usuário controlar os condicionadores de ar conectados à aplicação, efetuando as configurações oferecidas (ligar e desligar, alterar temperatura, e ligar e desligar o modo swing), visualizando o estado atual das configurações do condicionador de ar. O usuário poderá em uma única tela visualizar todos os condicionadores de ar de um determinado cômodo.

Constraints
Pré-condição. O usuário esteja autenticado à aplicação.

Pré-condição. Os condicionadores de ar estão conectados ao Arduino e configurados por cômodos.
Pós-condição. O condicionador de ar deve assumir as configurações determinadas.
Pós-condição. As configurações determinadas devem ser mantidas na tela respectiva ao condicionador de ar.

Cenários
Acessar condicionador de ar {Principal}.
1. Sistema apresenta na tela todos os cômodos da casa.
2. O usuário seleciona o cômodo desejado para a operação.
3. A aplicação apresenta todos os condicionadores de ar configurados para o cômodo.
4. O usuário seleciona o condicionador de ar desejado para a operação.
5. A aplicação apresenta o estado atual do condicionador de ar e todas as configurações possíveis.

Ligar condicionador de ar {Alternativo}.
No passo 5, se o condicionador de ar está desligado.
5.1. O usuário pressiona o botão para ligar condicionador de ar.
5.2. A aplicação liga o condicionador de ar pressionado.
5.3. A aplicação grava a operação na base de dados histórica.

Desligar condicionador de ar {Alternativo}.
No passo 5, o condicionador de ar está ligado.
5.1. O usuário pressiona o botão para desligar condicionador de ar.
5.2. A aplicação desliga o condicionador de ar pressionado.
5.3. A aplicação grava a operação na base de dados histórica.

Alterar temperatura do condicionador de ar {Alternativo}.
No passo 5, o condicionador de ar está ligado.
5.1. O usuário seleciona a temperatura desejada para o condicionador de ar.
5.2. A aplicação altera a temperatura do condicionador de ar de acordo com o selecionado.
5.3. A aplicação grava a operação na base de dados histórica.

Alterar movimentação das aletas para swing {Alternativo}.
No passo 5, o condicionador de ar está ligado.
5.1. O usuário altera a movimentação das aletas do condicionador de ar para o modo swing.
5.2. A aplicação altera a movimentação das aletas do condicionador de ar para o modo swing.
5.3. A aplicação grava a operação na base de dados histórica.

UC05 - Visualizar situação das janelas
Permite ao usuário visualizar a situação das janelas conectadas à aplicação. O usuário poderá em uma única tela visualizar todas as janelas existentes em um determinado cômodo.

Constraints

Pré-condição. O usuário esteja autenticado à aplicação.
Pré-condição. As janelas estão conectadas ao Arduino e configuradas por cômodos.

Cenários
Consulta serviço {Principal}.
1. Sistema apresenta na tela todos os cômodos da casa.
2. O usuário seleciona o cômodo desejado para a operação.
3. A aplicação apresenta todas as janelas configuradas para o cômodo com sua respectiva situação (aberta ou fechada)

APÊNDICE B – Descrição do Dicionário de Dados

Este Apêndice apresenta a descrição das tabelas do banco de dados apresentadas na seção de especificação deste trabalho. Nos Quadros de 7 a 14 estão o dicionário de dados das tabelas da aplicação. Os tipos de dados utilizados nos atributos são:

a) *int*: armazena números inteiros;
b) *varchar*: armazena caracteres alfanuméricos;
c) *date*: armazena data e hora;
d) *bit*: armazena números binários, utilizado para campos booleanos;
e) *char*: corresponde a caracteres.

Quadro 7 – Tabela de residências

Home					
Armazena as informações de conexão a residência.					
Campo	Descrição	Tipo	Tamanho	Chave Primária	Chave Estrangeira
Id	Código único de identificação.	*VARCHAR*	255	Sim	Não
Ip	Endereço IP da casa utilizado para conectar-se a esta.	*VARCHAR*	255	Não	Não

Quadro 8 – Tabela de cômodos

Room					
Armazena os nomes dos cômodos de uma determinada residência.					
Campo	Descrição	Tipo	Tamanho	Chave Primária	Chave Estrangeira
Id	Código único de identificação.	*INT*	11	Sim	Não
Name	Nome do cômodo.	*VARCHAR*	255	Não	Não
Home_id	Código da casa a qual o cômodo pertence.	*VARCHAR*	255	Não	Sim

Quadro 9 – Tabela de equipamentos

Device					
Armazena as informações dos equipamentos de um determinado cômodo.					
Campo	Descrição	Tipo	Tamanho	Chave Primária	Chave Estrangeira
Id	Código único de identificação.	*INT*	11	Sim	Não
ActionPort	Número da porta do Arduino usada para executar as ações no equipamento.	*INT*	11	Não	Não
Name	Nome do equipamento.	*VARCHAR*	255	Não	Não
StatusDevice	Situação atual do equipamento.	*INT*	11	Não	Não
Type	Tipo do equipamento (Luz, ar, janelas).	*CHAR*	1	Não	Não
Room_id	Código do cômodo ao qual o equipamento pertence.	*INT*	11	Não	Sim

Quadro 10 – Tabela de condicionadores de ar

AirConditioner					
Armazena as informações dos condicionadores de ar.					
Campo	Descrição	Tipo	Tamanho	Chave Primária	Chave Estrangeira
Id	Código único de identificação.	*INT*	11	Sim	Não
Ligado	Identificador do estado do ar (ligado ou desligado).	BIT	1	Não	Não
Temperatura	Temperatura atual do ar condicionado.	*INT*	11	Não	Não
Device_id	Código do equipamento ao qual o ar condicionado é referente.	*INT*	11	Não	Sim

Quadro 11 – Tabela de usuários

IUser					
Armazena as informações dos usuários.					
Campo	Descrição	Tipo	Tamanho	Chave Primária	Chave Estrangeira
Id	Código único de identificação.	*INT*	11	Sim	Não
Email	*Email* do usuário.	*VARCHAR*	255	Não	Não
Name	Nome do usuário.	*VARCHAR*	255	Não	Não
Sex	Sexo do usuário.	*VARCHAR*	255	Não	Não
Home_id	Código da residência do usuário.	*VARCHAR*	255	Não	Sim

Quadro 12 – Tabela de ações

IAction					
Armazena as informações das ações da aplicação.					
Campo	Descrição	Tipo	Tamanho	Chave Primária	Chave Estrangeira
Id	Código único de identificação.	*INT*	11	Sim	Não
Nome	Nome da ação.	*VARCHAR*	255	Não	Não
Device_id	Código do equipamento ao qual a ação é referente.	*INT*	11	Não	Sim

Quadro 13 – Tabela de Histórico de ações

HistAction					
Armazena o histórico das ações de uma determinada residência.					
Campo	Descrição	Tipo	Tamanho	Chave Primária	Chave Estrangeira
Id	Código único de identificação.	*INT*	11	Sim	Não
DateTime	Data e hora em que a ação ocorreu.	*DATE*		Não	Não
Action_id	Código da ação executada.	*INT*	11	Não	Sim
Home_id	Código da residência onde a ação foi executada.	*VARCHAR*	255	Não	Sim

Quadro 14 – Tabela de ações programadas para execução

ToDoAction					
Armazena o histórico das ações de uma determinada residência.					
Campo	Descrição	Tipo	Tamanho	Chave Primária	Chave Estrangeira
Id	Código único de identificação.	*INT*	11	Sim	Não
Activated	Identificador se a ação está ativada.	*BIT*	1	Não	Não
DateTime	Data e hora em que a ação será executada quando ativa.	*DATE*		Não	Não
Action_id	Código da ação que será executada.	*INT*	11	Não	Sim
Home_id	Código da residência onde a ação será executada.	*VARCHAR*	255	Não	Sim

ANEXO A – Circuito para controle da residência

Neste anexo apresenta-se através da Figura 38 o circuito utilizado para controlar as residências e o seu esquema elétrico na Figura 39. Para o desenho deste circuito e do esquema foi utilizada o programa Fritzing.

Figura 38 – Circuito para controle da residência

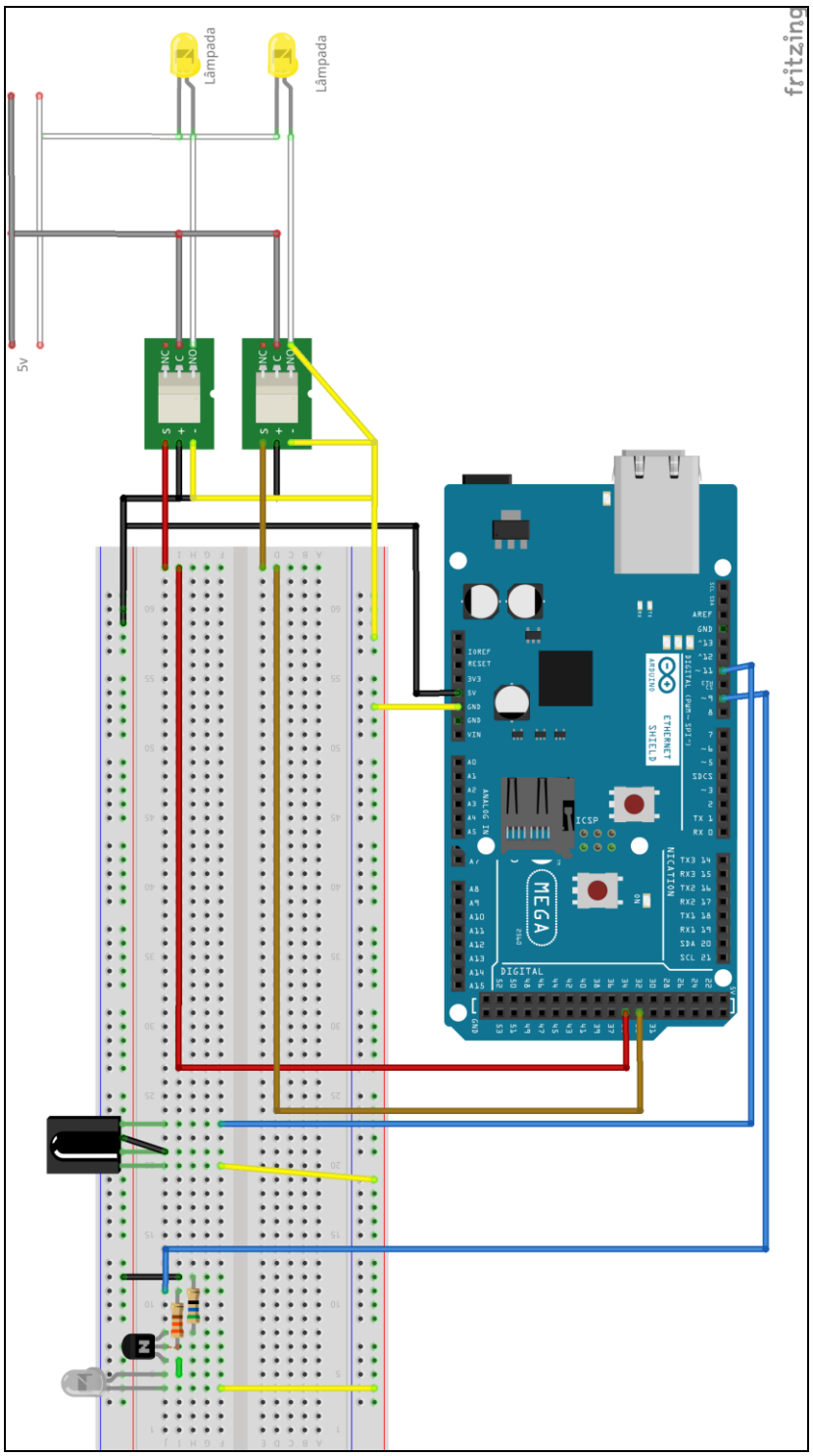


Figura 39 – Esquema elétrico do circuito para controle da residência