UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Automação Residencial de baixo custo por meio de dispositivos móveis com sistema operacional Android

Autor: Lucas Bragazza Beghini

Orientador: Prof. Dr. Evandro L. L. Rodrigues

São Carlos
2013

Lucas Bragazza Beghini

Automação residencial de baixo custo por meio de dispositivos móveis com sistema operacional Android

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

Curso de Engenharia Elétrica com ênfase
em Eletrônica

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2013

Beghini, Lucas Bragazza
B416a Automação residencial de baixo custo por meio de dispositivos móveis com sistema operacional Android / Lucas Bragazza Beghini; orientador Evandro Luis Linhari Rodrigues. São Carlos, .

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, .

1. Automação residencial. 2. Android. 3. Arduino. 4. Ethernet Shield. 5. App Inventor. I. Titulo.

FOLHA DE APROVAÇÃO

Nome: Lucas Bragazza Beghini

Título: "Automação residencial de baixo custo por meio de dispositivos móveis com sistema operacional Android"

Trabalho de Conclusão de Curso defendido e aprovado em 28 / 11 / 2013,

com NOTA 10,0 (Dez,), pela Comissão Julgadora:

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

Prof. Dr. Danilo Hernane Spatti - SEL/EESC/USP

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

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

Agradecimentos

A Deus, por sempre me dar forças para superar os obstáculos encontrados.

A meus pais, pelo apoio incondicional em todos os momentos, e por terem fornecido todo o suporte necessário para minha formação.

A todos os meus familiares, em especial à minha tia Lucia, pelos conselhos e auxílio na revisão dos textos, e a meu tio Bruno, pelas discussões sempre pertinentes sobre o tema e pelo empréstimo de sua câmera IP pessoal.

À minha namorada, Samara, por todo o carinho e pelo auxílio com a revisão de textos e formatação desta monografia.

A meu orientador, Evandro, pelos conselhos, cobranças e ensinamentos, e por sua competência em ministrar a matéria "Aplicação de Microprocessadores II", que foi de fundamental importância para realização deste trabalho.

A meus amigos, por tornarem esses 5 anos de graduação ainda mais especiais.

A todos vocês, o meu sincero obrigado.

Resumo

O objetivo deste trabalho foi a implementação de um sistema de automação residencial de baixo custo, utilizando um *Arduino Uno* como central de automação, com acesso via internet de qualquer lugar mundo. Foi desenvolvido um aplicativo para celulares com sistema operacional *Android*, capaz de controlar alguns processos de uma residência de acordo com as necessidades do usuário, tais como: controle de alimentação de animais de estimação, sistemas de iluminação e alarme. Para criação do aplicativo foi utilizado o *App Inventor,* que possui uma interface gráfica de programação, possibilitando aos usuários sem experiência com programação em linguagem *Java*, o desenvolvimento de aplicativos. O binômio custo-benefício foi alcançado indicando que o valor agregado dos benefícios aos usuários tais como: praticidade, segurança e simplicidade, na execução das tarefas com baixo custo de investimento, seja o principal estímulo para investimentos em melhorias na automação residencial.

Palavras-chave: Automação Residencial, Arduino, Ethernet Shield, Android, App Inventor

Abstract

The objective of this work was the implementation of a low cost home automation system, using an Arduino Uno as central automation, with access through the Internet from anywhere in the world. It was developed an application for mobile phones with Android operating system, capable to control some processes of a residence according to user needs, such as :control of feeding pets, lighting and alarm systems. To create the application was used App Inventor, which has a graphical interface programming, enabling users without programming experience in Java language, application development. The cost-benefit was achieved indicating that the value of user benefits such as convenience, security and simplicity, in performing the tasks with low investment cost, be the main stimulus for investments and improvements in home automation systems.

Keywords: Home Automation, Arduino, Ethernet Shield, Android, App Inventor

Lista de Figuras

Lista de Siglas

CLPs (Controladores Lógicos Programáveis)

PC (Personal Computer)

LED (Light Emitting Diode)

LCD (Liquid Crystal Display)

KB (Kilobyte)

ARM (Advanced RISC Machine)

SRAM (Static Random Access Memory)

SPI (Serial Peripheral Interface)

TCP/IP – (Transmission Control Protocol) / IP (Internet Protocol)

SMTP – (Simple Mail Transfer Protocol)

FTP – (File Transfer Protocol)

HTTP - (HyperText Transfer Protocol)

DNS (Domain Name System)

gTLDs (Generic Top Level Domains)

ccTLDs (Country Code Top Level Domains)

DDNS (Dynamic Domain Name System)

DHCP (Dynamic Host Configuration Protocol)

MIT (Massachusetts Institute of Technology)

USB (Universal Serial Bus)

PIR (Passive Infrared)

PIC (Peripherical Interface Controller)

SMS (Short Message Service)

IDE (Integrated Development Environment)

SMTP (Simple Mail Transfer Protocol)

TLS (Transport Layer Security)

SSL (Security Sockets Layer)

GSM (Global System for Mobile Communications)

Sumário

1.Introdução

A automação residencial, também conhecida como domótica, corresponde a utilização das inovações tecnológicas para satisfazer as necessidades e, principalmente, o conforto dos integrantes de determinada habitação. A palavra domótica tem origem na palavra latina "Domus" que significa "Casa", unida a palavra "Robótica", que é a automatização e controle de qualquer processo. A área está em crescente evolução nas últimas décadas, auxilada pelo avanço da tecnologia e aproximação da mesma com atividades ligadas ao cotidiano. Tem como principal origem a automação industrial, enriquecida com o surgimento dos CLPs (Controladores Lógicos Programáveis) durante a década de 60. Dessa maneira, a domótica permite ao usuário controlar dispositivos eletrônicos de sua residência através de interfaces de controle (EUZÉBIO, M. V.M. & MELLO, E. R., 2013).

Algumas empresas de tecnologia buscaram transferir o crescente desenvolvimento da microeletrônica para aplicações em residências de modo a oferecer soluções para automatização de processos afetos a área. Contudo, perceberam que para a implementação da automação residencial é imprescindível que todo o sistema funcione com bom nível de robustez, principalmente com relação a segurança, além de oferecer uma interface simples e objetiva de comunicação com o usuário, considerando que este não possui necessariamente conhecimento técnico para operar um sistema mais complexo. Devido a estes motivos, a área da automação residencial não acompanhou a evolução da automação industrial, que acabou desenvolvendo-se mais rapidamente. Segundo Bortoluzzi (2013), "a década de 70 pode ser considerada o marco inicial da automação residencial, quando foram lançados nos EUA os primeiros módulos inteligentes chamados X-10". O protocolo X-10 foi desenvolvido para controle remoto de dispositivos utilizando a própria rede elétrica como canal de comunicação. Particularmente, esta era uma característica interessante do sistema, pois permitia o controle de dispositivos remotos sem que fosse necessária uma alteração na infraestrutura elétrica da residência.

Mais adiante, na década de 80, com a popularização dos computadores pessoais (PCs), pôde-se pensar em um PC como central de automação. Entretanto, a grande desvantagem desse sistema é o elevado consumo, devido a necessidade de manter o PC sempre ligado. A partir desse problema, partiu-se para o desenvolvimento de dispositivos embarcados que os substituíssem, através da utilização de microprocessadores e microcontroladores.

Desde então, foram sendo incorporados alguns meios de comunicação com a central de automação, constituída por um microcontrolador atuando em um sistema embarcado. Com a popularização da internet de banda larga, essa tecnologia passou a ser altamente explorada, possibilitando também técnicas de monitoramento não só presenciais, mas também à distância, sendo possível dessa forma controlar a residência através de uma *Web page* (CRUZ,2009). Outros meios, como por exemplo o *Bluetooth*, padrão de comunicação desenvolvido para integração entre celulares e periféricos, também foram incorporados, com o objetivo de, por exemplo, controlar lâmpadas à pequena distância (SILVA, B.C.R & CÂNDIDO, L.A.A, 2011).

Atualmente é possível compor sistemas para automação residencial com bom nível de controle, intermediado por sistemas embarcados cada vez mais poderosos, com acesso remoto via internet ou via sistema de telefonia móvel. Com a crescente popularização de *smartphones*, é cada vez mais estimulante a criação de aplicativos, através de ferramentas acessíveis ao usuário, para se controlar os processos residenciais sem a necessidade da utilização de um navegador para abertura de uma *Web page*.

Além disso, a integração entre os processos a serem controlados e o aproveitamento total da central de automação, em questão de capacidade do sistema, são fatores importantes a serem considerados, visto que é necessário apenas uma central para controlar várias aplicações, o que determina um melhor custo-benefício do que implementá-las separadamente.

Neste contexto, a proposta deste trabalho é o controle de alguns processos residenciais escolhidos que estão presentes no dia-a-dia dos habitantes de uma residência, como por exemplo: controle de lâmpadas, alarme e alimentação de animais de estimação, tudo por meio de um *smartphone*.

O controle de lâmpadas tem como principal objetivo satisfazer o conforto dos integrantes de determinada habitação, eliminando a necessidade de se utilizar o interruptor para tal tarefa. Ademais, pode ser utilizado também por questões de segurança, acendendo a lâmpada em determinada ocasião onde essa ação se faça necessária, principalmente quando o morador se encontra ausente.

Já o sistema de alarme para segurança de residências vem ganhando bastante mercado nos últimos tempos devido ao aumento da violência, principalmente nos grandes centros urbanos. Desta forma, este trabalho propõe o desenvolvimento de um sistema de alarme residencial de baixo custo, funcionando em conjunto com as outras

aplicações, e que informe ao usuário, imediatamente, em seu celular, quando houver algum movimento suspeito no momento em que o alarme estiver ativado, além de disparar uma sirene de segurança.Também é possível ativar ou desativar este mesmo alarme de qualquer lugar do mundo, via internet.

A terceira e última aplicação desenvolvida para esta monografia é um sistema de alimentação canina, ou de qualquer outro animal de estimação, via internet. A motivação para o desenvolvimento desse sistema é permitir ao dono do animal em questão, monitorá-lo a distância e alimentá-lo de acordo com a necessidade após a verificação, em tempo real, da quantidade de comida que ainda resta no recipiente através de uma câmera.

Em Março do ano de 2012 o americano Nat Morris criou um mecanismo de alimentação canina via internet por intermédio do *Twitter* (DAILYMAIL,2012). O sistema era composto por um microprocessador *Nanode* (um microprocessador de código aberto com conexão com a internet integrada, da mesma família do *Arduino* e que utiliza também o *ATMega328* como microcontrolador) e um motor de passo de uma impressora *HP Deskjet 500*. Para que o cão fosse alimentado era necessário apenas enviar um comando para a conta @FeedToby no*Twitter.* Porém, como tal comando poderia ser enviado por qualquer pessoa, rapidamente o americano passou a ter problemas com quantidades ilimitadas de comida, o que o obrigou a limitar as porções em duas vezes ao dia. Há também no mercado alimentadores baseados em temporizadores, que, de tempo em tempo, alimentam o animal. Eles tem como desvantagem a ausência de um sistema de monitoramento que informe o usuário se há comida disponível, ou se o cão está de fato se alimentando, além de serem alternativas de alto custo.

Portanto, a idéia deste trabalho é integrar o módulo de alimentação canina à central de automação, funcionando em conjunto com as outras aplicações, e acionado por meio de um aplicativo próprio, controlado somente pelo seu dono e monitorado por uma câmera que se mantém todo o tempo conectada à internet. A aplicação citada, no entanto, não foi construída, foi desenvolvida apenas a idéia para a solução.

1.1 Objetivos

Aproveitando a popularização dos *smartphones* e *tablets* e as facilidades a eles agregadas, como acesso a internet e suporte de aplicativos, o objetivo do projeto é o controle de processos residenciais, tais como: iluminação, alarme e alimentação de animais, com baixo custo de implementação, por meio de dispositivos móveis que utilizem o sistema operacional *Android*. O sistema em questão foi desenvolvido pela *Google* e tem como principal vantagem o acesso livre do usuário a hardware e software, permitindo a criação de aplicativos que atendam a uma necessidade específica. Tal característica foi aproveitada no projeto com a criação de um aplicativo, através de uma ferramenta de fácil acesso ao usuário, o *App Inventor*, que, em comunicação com uma plataforma de hardware e atuadores, controle determinados processos de uma residência, de acordo com as necessidades do habitante.

2. Embasamento Teórico

Neste capítulo será apresentada toda a fundamentação teórica necessária para o desenvolvimento desta monografia, bem como alguns componentes que foram utilizados e que merecem um maior aprofundamento teórico a respeito de sua composição e funcionamento. O diagrama de blocos do sistema completo pode ser verificado na Figura 2.1.

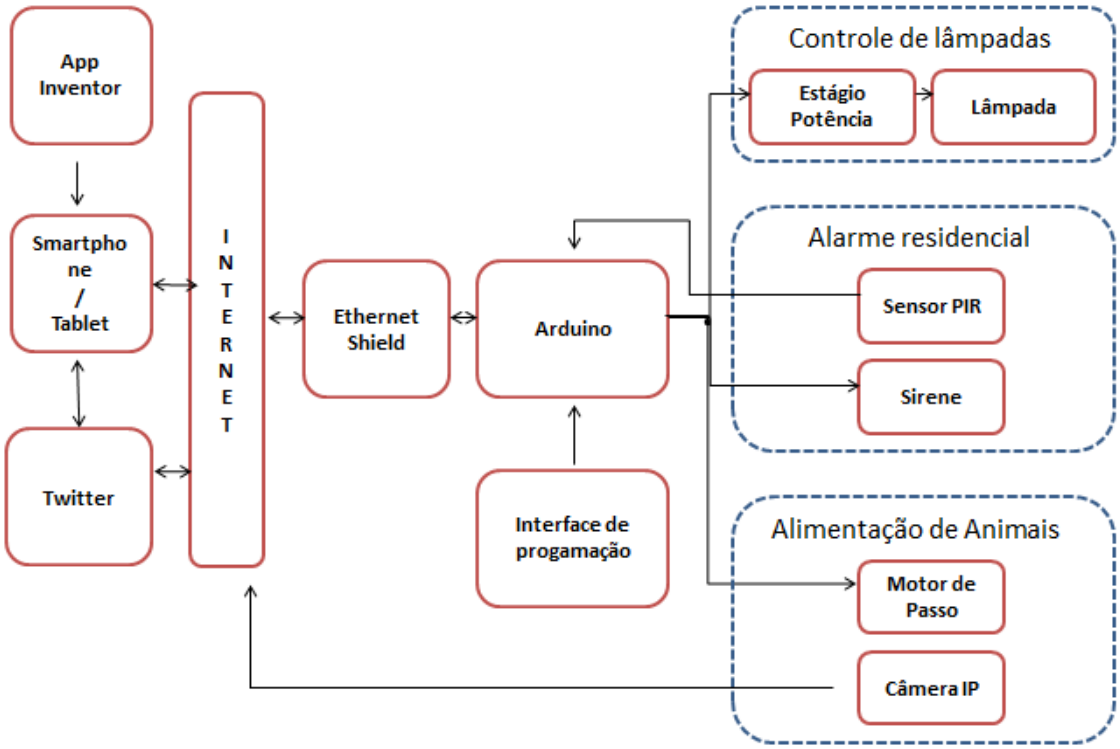


Figura 2.1 - O projeto

2.1 Arduino

Segundo McRoberts (2011,p.22), "O Arduino é o que chamamos de plataforma de computação física ou embarcada, ou seja, um sistema que pode interagir com seu ambiente por meio de hardware e software."

Graças a esta característica, sua aplicabilidade no mundo eletrônico é muito vasta, sendo possível o controle de uma série de dispositivos, os quais podemos citar: sensores, motores elétricos, LEDS, displays LCD, chaveamento de transistores, dentre outros.

O *Arduino* surgiu em 2005, na Itália, criado por um professor chamado Massimo Banzi, que desejava ensinar a seus alunos um pouco de eletrônica e programação de dispositivos. Como seus alunos eram de um curso de Design, ensiná-los eletrônica sem uma base construída não era uma tarefa simples. A inexistência de algo barato no mercado e que tivesse ferramentas poderosas, também dificultavam as idéias de Massimo. Devido a esses fatores, o professor, com auxílio de David Cuartielles, decidiram criar sua própria placa, com a ajuda do aluno de Massimo, David Mellis, que ficou responsável por criar a linguagem de programação do *Arduino* (BOEIRA, 2013). Assim, apareceu para o mundo uma das mais populares aplicações de eletrônica de hoje em dia e que tem se espalhado rapidamente pelo planeta.

O fato que mais chama atenção em relação ao *Arduino* é a liberdade dada ao usuário. Tanto o software quanto o hardware são livres, de modo que qualquer pessoa tenha acesso. Inclusive qualquer usuário pode desenvolver seu próprio Arduino,com a ressalva de que não use este mesmo nome em sua placa de circuito impresso.

A Figura 2.2 mostra o *Arduino Uno*, um dos modelos mais simples de *Arduino* existentes no mercado e utilizado neste trabalho.

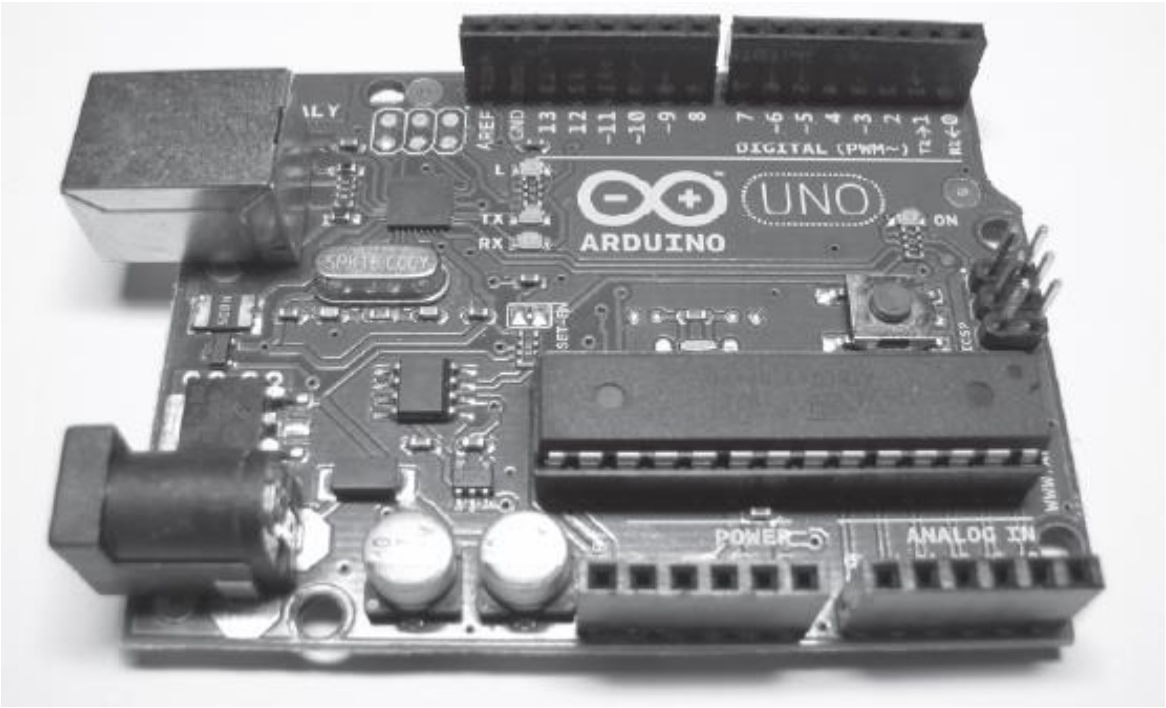


Figura 2.2 - Arduino Uno (McROBERTS,2011)

As características do sistema são as seguintes, seguindo o *site* oficial do produto (ARDUINO-1, 2013).

-**Microcontrolador**: ATmega328

-**Tensão de operação**: 5V

-**Tensão de entrada(recomendada)**: 7-12V

-**Tensão de entrada (limites):** 6-20V

-**Pinos de entrada/saída digitais**: 14 (6 podem fornecer saída PWM (Modulação por Largura de Pulso))

-**Pinos de entrada analogica:** 6

-**Corrente DC por pino de E/S**: 40 mA

-**Memória Flash**: 32 KB

-**SRAM**: 2 KB

-**EEPROM:** 1 KB

-**Frequência de clock** : 16 MHz

Embora seja limitado para aplicações consideradas de grande porte, devido a pouca capacidade de processamento e o baixo número de portas digitais e analógicas disponíveis, o *Arduino Uno* ainda é uma ferramenta poderosa para vários tipos de aplicações, além de se tratar de uma opção viável no mercado em termos financeiros.

2.2 Ethernet Shield

Outra vantagem do *Arduino* é a existência de vários *shields* que permitem ao usuário estender a capacidade do sistema ou especificar uma aplicação desejada. Os *shields* são placas de circuito impresso que são encaixados à placa principal e cumprem função específica no sistema.

Dentro desse contexto, o *Ethernet Shield*, compatível com *Arduino Uno*, é responsável por fazer a conexão do *Arduino* com a internet, através de um cabo de rede. Segundo o *site* oficial (ARDUINO-2, 2013), a forma de comunicação com a placa principal é feita utilizando o barramento *SPI* (*Serial Peripheral Interface*), através dos pinos 10, 11, 12 e 13. No pino 10 é feita a seleção do W5100, chip da *WIZnet*. Ele

fornece o protocolo *TCP/IP* para o *Arduino* na rede, possibilitando toda a comunicação com outro dispositivo via internet (para mais detalhes ver datasheet no Anexo A).

O protocolo TCP/IP é o principal protocolo de envio e recebimento de dados via internet. Por se tratar na verdade de um conjunto de protocolos integrados, pode também ser conhecido como "Pilha de Protocolos". É dividido em 4 camadas distintas, de forma a garantir a integridade dos dados que trafegam pela rede (TECHMUNDO, 2013). São as seguintes:

Camada de Aplicação: Trata-se da camada mais próxima ao usuário. Essa camada é utilizada para enviar ou receber informações de outros programas através da rede. Nesta mesma, é possível encontrar outros tipos de protocolos como SMTP (para email), FTP (transferência de arquivos) e o mais conhecido, HTTP (para navegar na internet). Uma vez que os dados tenham sido processados pela camada de aplicação, eles são enviados para a camada de transporte.

Camada de transporte: Tem por função principal receber os dados provenientes da camada anterior e dividi-los em blocos de dados, também conhecidos como pacotes.

Camada de rede: Feita a divisão, os dados empacotados são recebidos e anexados ao endereço virtual (IP) do dispositivo remetente e do destinatário.

Camada de Interface: Tem por função especificar os detalhes de como os dados são enviados fisicamente pela rede.Os protocolos utilizados nessa camada dependem do tipo de rede que está sendo utilizada. O tipo mais comum utilizado atualmente é o *Ethernet.*

Em suma, estes são os processos que ocorrem no *W5100* para se receber ou enviar um dado via internet.

Para montagem do sistema, basta encaixar o *Ethernet Shield* no *Arduino Uno*, nos terminais correspondentes, e ligar o cabo de rede proveniente do roteador na entrada *RJ45* do shield. A Figura 2.3 ilustra o sistema já montado:

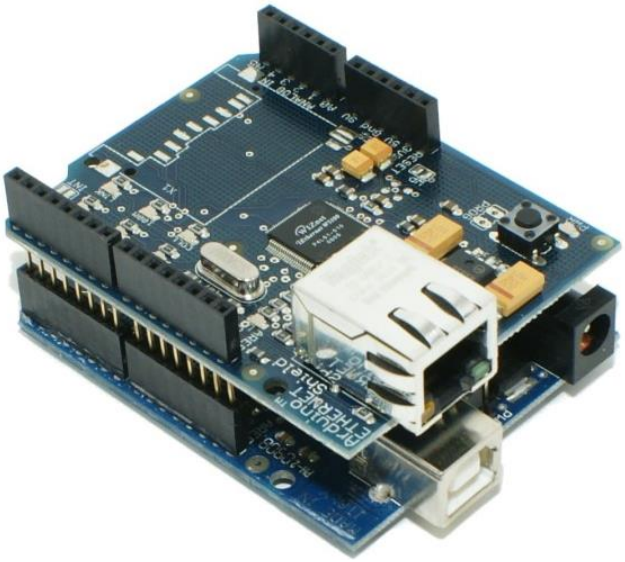


Figura 2.3 - Arduino Uno e Ethernet Shield (ARDUINOECIA,2013)

Pode-se observar na Figura 2.3 o *Ethernet Shield* encaixado sobre a placa de *Arduino Uno*. Cabe ressaltar que este encaixe, conforme visto anteriormente, acaba ocupando 4 pinos digitais do *Arduino*, diminuindo para 10 portas digitais disponíveis a capacidade do sistema, no caso do *Arduino Uno*.

O restante do processo é feito via software. É necessário a inclusão da biblioteca *Ethernet* no código e a configuração do IP que o *Arduino* terá na rede.

2.3 Interface de Programação

Mais uma vantagem do *Arduino* é a interface amigável de programação e comunicação com o microcontrolador. A conexão é feita do computador com o *Arduino* via USB, permitindo o *upload* de programas para o *AtMega328*. A interface disponível pode ser verificada na Figura 2.4.

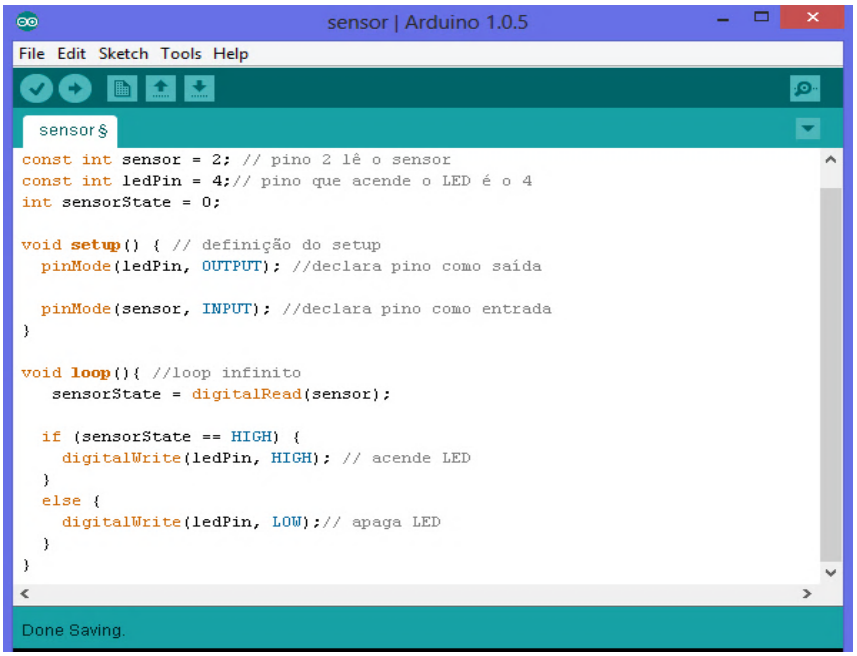


Figura 2.4 - Interface de Programação

A interface da Figura 2.4, além das características já citadas, permite a compilação do programa, inclusão de bibliotecas e alguns exemplos de aplicações. Ademais, também permite um monitoramento da comunicação serial (*Serial Monitor*), muito útil para identificação de erros no código, observação do comportamento do sistema e até mesmo envio de comandos por este meio em determinada aplicação. A linguagem de programação utilizada é a *linguagem C*.

2.4 Acesso externo ao Arduino

A forma de acesso externo ao *Arduino* não é um processo simples e requer alguns conhecimentos que serão expostos a seguir.

2.4.1 Intranet e Internet

Intranet é uma rede privada de uso exclusivo em determinado local, ou seja, de acesso restrito. Pode ser usada em empresas e em residências. Constituem a tradicional rede local e portanto não permite acesso externo.

Internet é a rede mundial de computadores e portanto um conglomerado de redes locais. Não possui acesso restrito e é possível acessá-la externamente de qualquer lugar do mundo.

Portanto, acessar qualquer dispositivo pela intranet ou pela internet são processos totalmente distintos. A Figura 2.5 ilustra o processo.

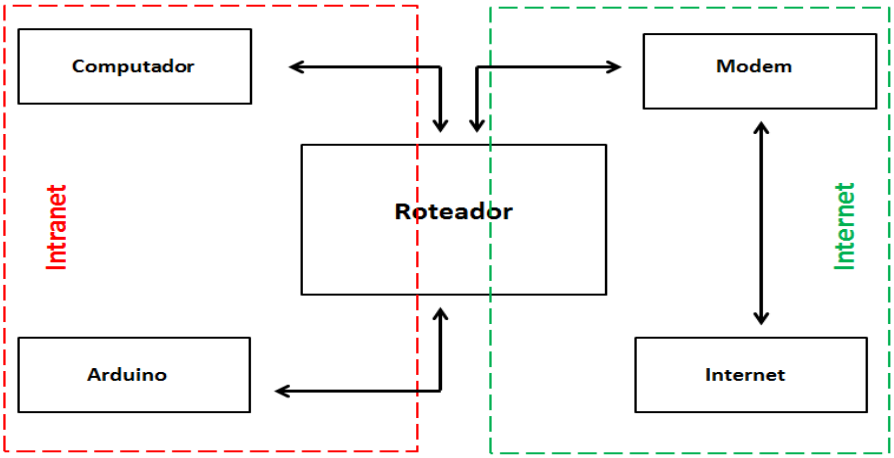


Figura 2.5 - Intranet e Internet

O acesso pela Intranet é exemplificado na cor vermelha, ou seja, a partir de um computador ou dispositivo móvel, acessa-se a rede local conectando-se ao roteador, que por sua vez envia o comando ao *Arduino.*

Já o exemplo destacado na cor verde, é um acesso pela Internet. De qualquer lugar do mundo é possível acessar o roteador, que por sua vez envia os comandos dados ao *Arduino*. Entretanto, é necessário saber qual IP o roteador e o servidor ou domínio ligado a ele possui na rede. Esta será uma questão abordada nos próximos tópicos.

2.4.2 DNS e DDNS

O serviço DNS (*Domain Name System*) é utilizado para a conversão de endereços IP a partir de nomes de domínio que possuem formato numérico. Ele elimina a necessidade de se ter que lembrar um determinado número para o endereço de um *site*, basta lembrar o endereço tradicional, com letras do alfabeto. Em outras palavras, há duas formas de acessar uma página na internet: pelo nome de domínio ou pelo endereço IP dos servidores nos quais ela está hospedada. Para tornar

desnecessária a digitalização da sequência numérica no navegador sempre que se quiser visitar um *site*, o DNS realiza a tarefa de traduzir as palavras que compõem a *URL* para o endereço IP do servidor, direcionando assim, o usuário para o local desejado.

A forma de se implementar este serviço ocorre por meio de vários bancos de dados espalhados ao redor do mundo, de forma que, quando se digita algum endereço no navegador, solicita-se aos servidores de DNS do respectivo provedor de internet (ou outros que se tenha especificado) que encontre o endereço IP associado ao referido domínio. Caso estes servidores não tenham esta informação, eles se comunicam com outros que possam ter.

Com o objetivo de tornar esse processo de busca mais prático, foram criados níveis hierárquicos para os domínios. O primeiro nível, e mais importante, é o chamado *Servidor Raiz (Root Server).* A sua função é responder diretamente às requisições de registros da zona raiz, retornando uma lista dos servidores de nome designados para o domínio apropriado. Atualmente, existem 13 servidores deste tipo no mundo inteiro, sendo que 10 deles estão nos Estados Unidos (2 na Europa e 1 na Ásia). Em caso de falha de algum deles, os outros garantem a integridade do sistema.

A hierarquia é seguida com domínios que apresentam extensões grafadas como: *.com*, *.net*, *.org, .info, .edu* e outros. Estas são chamadas de *gTLDs (Generic Top Level Domains)*.Há também terminações relacionadas a países, chamadas de *ccTLDs (Country Code Top Level Domains*)I, como por exemplo: *.br* para o Brasil, *.ar* para a Argentina, *.fr* para a França e assim por diante (INFOWESTER,2013). A Figura 2.6 ilustra o nível hierárquico mencionado e o exemplo da posição no organograma do *site* (REGISTRO, 2013).

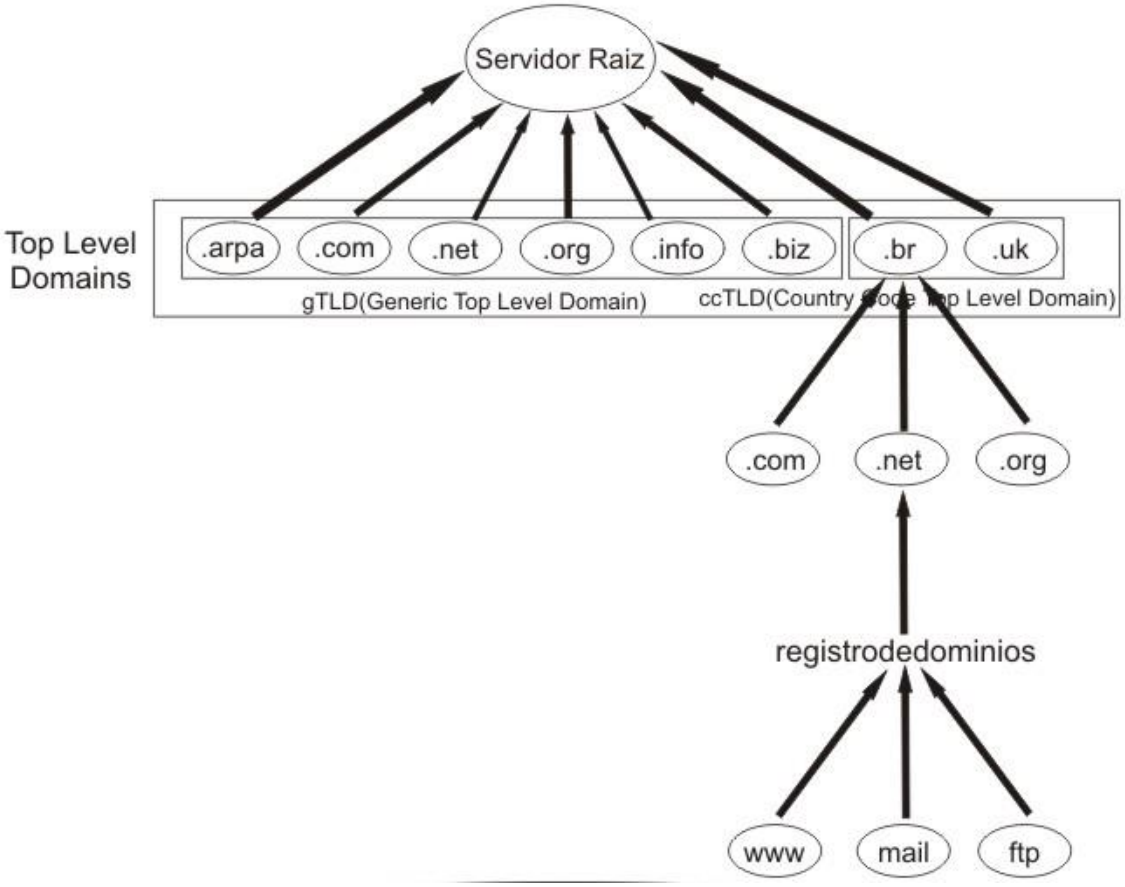


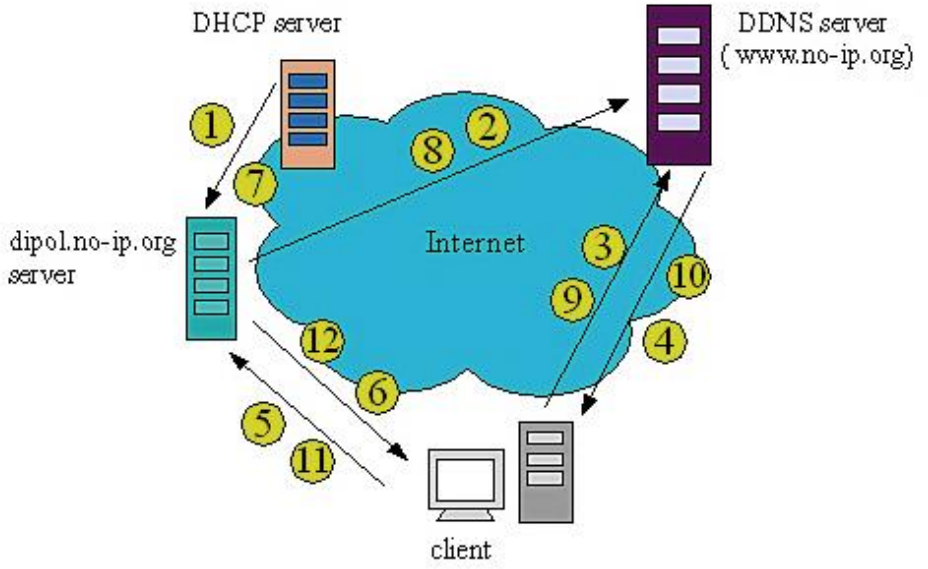
Figura 2.6 - Hierarquia de domínios (REGISTRO, 2013)

Dessa forma, caso o serviço de DNS do provedor de internet não encontre o domínio desejado, ele direcionará a busca a um *Servidor Raiz* que, por sua vez, direcionará ao nível hierárquico imediatamente inferior, e assim sucessivamente até encontrar o domínio especificado. O serviço do provedor de internet então armazena esse endereço em um *cache* de DNS. Assim não é necessário fazer uma nova busca caso solicitado novamente o mesmo domínio.

Um outro fator a ser considerado é que, geralmente, clientes de provedores de internet não possuem IP fixo. Essa é uma exclusividade de empresas,instituições de ensino ou de outras entidades que necessitam deste fator. O problema é que sem um IP fixo, é impossível o acesso externo (via internet), já que o servidor DNS aponta para um IP específico e se este mudar constantemente, os dados DNS deixam de ser válidos. A solução para este impasse é o uso do DDNS (*Dynamic Domain Name System*).

O funcionamento do DDNS ocorre da mesma forma que o servidor DNS,fornecendo um banco de dados contendo as relações entre o domínio e os endereços numéricos. A diferença é que agora este banco de dados pode ser atualizado a pedido do proprietário do domínio. O proprietário do domínio acessa o servidor DDNS que por sua vez envia informações do IP do domínio ao servidor DNS.

Analisando sob outro ponto de vista, o servidor DDNS é uma ponte entre o cliente e o domínio a ser acessado, no caso deste trabalho, gerado pelo *Arduino,* que viabiliza o acesso pela internet a esse dispositivo. A Figura 2.7, retirada do *site* (DIPOL, 2013), ilustra bem o processo.



1. Seu endereço IP é 83.17.30.134
2. dipol.no-ip.org é 83.17.30.134
3. Qual é o endereço IP atual do dipol.no-ip.org?
4. dipol.no-ip.org se agora 83.17.55.212
5. Oi, 83.17.30.134 me enviar o seu conteúdo www
6. Aqui está!
7. Seu endereço IP foi alterado para 83.17.55.212
8. dipol.no-ip.org se agora 83.17.55.212
9. Qual é o IP atual do dipol.no-ip.org?
10. dipol.no-ip.org se agora 83.17.55.212
11. Oi, 83.17.55.212. Envie-me seu conteúdo www.
12. Aqui está!

Figura 2.7 - Funcionamento do servidor DDNS (DIPOL,2013)

O DHCP (*Dynamic Host Configuration Protocol*) é um protocolo cliente/servidor que fornece automaticamente um *host* de IP com seu endereço IP e outras informações de configuração relacionados, como o *gateway* padrão e máscara de sub-

rede. Em redes domésticas, é configurado no roteador e fornece um endereço de IP exclusivo para acessar aquela rede específica.Neste trabalho, assim como no exemplo da Figura 2.6, o servidor DDNS usado é proveniente do *no-ip.*

O *no-ip* é um *site* onde é possível alocar servidores DDNS gratuitamente. Basta fazer o cadastro e configurar o servidor de maneira que atenda as necessidades do usuário. Feito isso, basta acessar no navegador o endereço configurado para o servidor criado no *site* e o tráfego será direcionado ao domínio gerado pela sua rede doméstica, que neste caso é contemplada pelo *Arduino* e o *Ethernet Shield.*

2.5 App Inventor

O *App Inventor* é uma ferramenta para criação de aplicativos que possui uma interface gráfica de programação, possibilitando ao usuário sem experiência com programação em linguagem *Java*, desenvolver aplicações para dispositivos com sistema operacional *Android.* Foi originalmente colocado a disposição pela *Google* em Dezembro de 2010 e repassada ao MIT (*Massachusetts Institute of Technology*), que é responsável por manter o sistema ativo desde Dezembro de 2011.

A interface utiliza a biblioteca *Java* de código aberto *Open Blocks* para criação de um ambiente visual de programação, semelhante a um diagrama de blocos. A biblioteca citada acima é distribuída pelo MIT e proveniente das teses de mestrado de Ricarose Roque, Professor Eric Klopfer e Daniel Wendel. O compilador que traduz a linguagem de blocos visual para aplicação em *Android* utiliza a estrutura de linguagem *Kawa* (linguagem de programação para plataforma *Java*), dentre outros dialetos da mesma (APPINVENTOR, 2013).

Seguindo Wolber et al. (2011) o objetivo do usuário é unir esses blocos em uma espécie de "quebra-cabeças", de modo a satisfazer sua necessidade de aplicação. Essa é uma característica importante do sistema, pois não permite a alocação de determinados blocos onde não seria possível alocá-los. A justificativa dos criadores para o desenvolvimento da ferramenta é a utilização do sistema para se engajar idéias poderosas através da aprendizagem rápida e contínua.

Embora pareça intuitivo, a utilização do *App Inventor* ainda requer conhecimentos básicos de lógica de programação e um estudo detalhado dos blocos que compõem o sistema, principalmente para aplicações com elevado grau de complexidade.

Para dar início a criação de um novo aplicativo basta acessar o *site* (APPINVENTOR) e clicar na aba "Invent". Os projetos ficam hospedados na nuvem e podem ser acessados de qualquer lugar, sendo apenas necessário fazer *login* em sua conta do *Google*. Na mesma página, pode-se ainda acessar tutoriais de desenvolvimento, guias de iniciação, além de um fórum entre os usuários, todos estes disponíveis somente na língua inglesa.

Após a criação de um novo projeto, a tela inicial de desenvolvimento aparece como na Figura 2.8. É nesta onde se começa a criação do aplicativo. É possível customizar o seu *app* de acordo com sua criatividade incluindo botões, imagens, cores, telas secundárias e outros recursos que desejar. Além disso, é necessário incluir elementos que não estão diretamente ligados ao *design* do aplicativo, mas que serão utilizados futuramente na montagem do diagrama de blocos do sistema, como por exemplo, para o caso deste trabalho, o componente "Web" (que fornece funções para HTTP GET) e o "TinyWebDb" (componente que se comunica com um servidor Web para armezenar,transferir ou recuperar informações). Só é possível tratar todos esses elementos não-visíveis na montagem do diagrama de blocos, se os mesmos tiverem sido incluídos nesta tela de desenvolvimento.

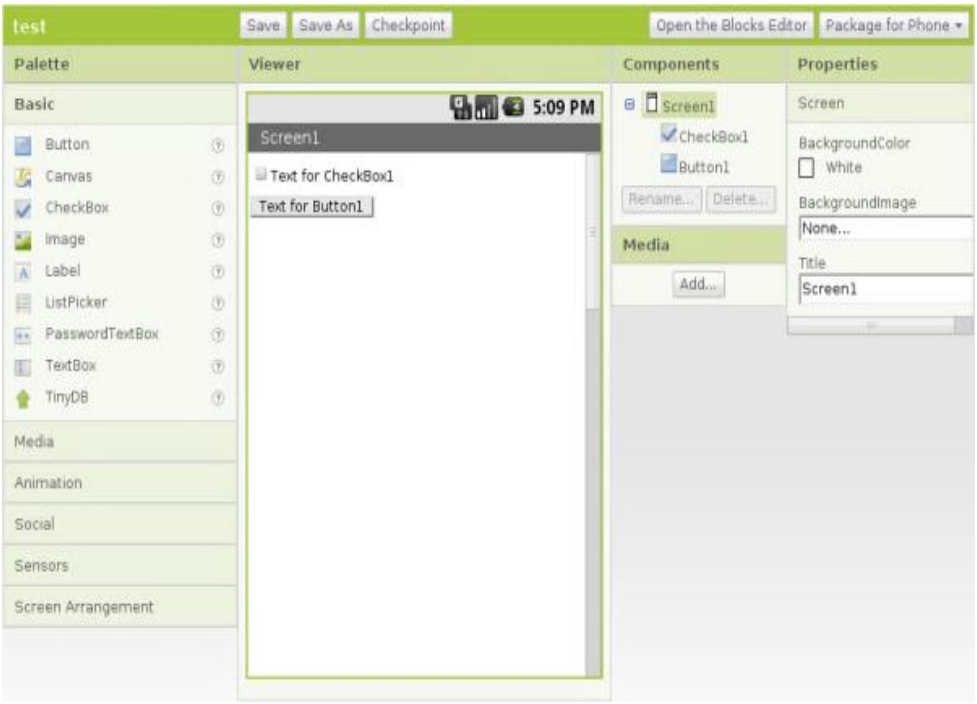


Figura 2.8 - Tela de desenvolvimento do aplicativo

Após desenvolvida a primeira parte é preciso construir o diagrama de blocos do aplicativo, através da aba "Open Blocks Editor". Cada tela de desenvolvimento criada, corresponde a uma aplicação que precisa ser desenvolvida no Editor de Blocos. É necessário que o programa Java esteja instalado no computador. A Figura 2.9 ilustra o ambiente de desenvolvimento.

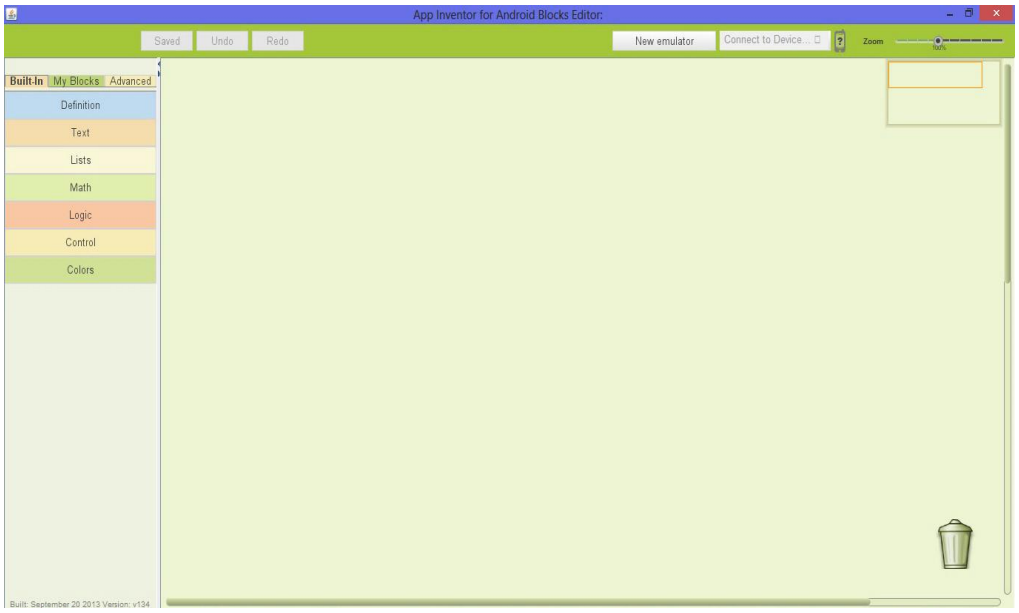


Figura 2.9 - Editor de blocos do *App Inventor*

Na aba "Built-in" encontram-se todos os elementos genéricos para criação da aplicação, separados por cores de acordo com o tipo de dado para facilitar o entendimento do usuário. Na aba "MyBlocks" estão todos os blocos referentes aos elementos que o usuário incluiu na tela inicial de desenvolvimento. Cabe ao usuário montar esses blocos, respeitando suas funcionalidades, para concluir a aplicação.

Após construída a aplicação, o usuário tem duas opções para testá-la. A primeira é a utilização do emulador disponível no próprio programa. A segunda é fazer a comunicação com o celular via USB. Neste último caso, é necessário fazer o download do aplicativo *MIT AICompanion,* disponível gratuitamente na *Play Store,* e pareá-lo, através de um código de segurança, com o computador.Uma vez estabelecida a comunicação, é possível alterar o aplicativo criado no computador e este será modificado em tempo real no celular, otimizando os processos de testes.

Realizados os testes, o usuário tem mais uma vez duas opções para transferir a aplicação definitivamente para seu *smartphone*. Ambas ocorrem através da aba "Package for Phone" na tela inicial de desenvolvimento (Figura 2.8). A primeira é mediante a utilização da porta USB do PC diretamente, sendo necessário, neste caso, estabelecer a conexão com o aplicativo *MIT AICompanion*, conforme descrito no parágrafo anterior. A outra é fazer o *download* para o computador do arquivo com extensão ".*apk"* e transferir ao *smartphone* via *Bluetooth*, *Email* ou qualquer outro meio de comunicação possível entre os dois. Cabe ressaltar que ao instalar no celular o aplicativo é possível que as configurações de segurança do celular o bloqueiem, visto que ele não é um aplicativo proveniente da *Play Store*. Neste caso, será necessário alterar as configurações de segurança do celular, permitindo a instalação de aplicativos que não são do *Android Market.*

De fato, o *App Inventor* é uma ferramenta poderosa para criação de aplicativos dos mais variados tipos. O editor de blocos contempla quase todas as características necessárias para criação de aplicativos dos mais simples aos mais complexos. Além disso, a interface amigável com o usuário constitui uma importante característica do sistema, principalmente para usuários sem experiência com programação em linguagem *Java*.

2.6 Sensor PIR

O *Sensor PIR (Passive Infrared Sensor)* é um sensor de movimento capaz de detectar níveis de radiação infravermelha. A radiação infravermelha existe no espectro eletromagnético com um comprimento de onda maior do que a luz visível ao ser humano. Portanto ela não pode ser vista, mas pode ser detectada. Objetos que geram calor também geram radiação infravermelha e a estes objetos pode-se incluir animais e o próprio corpo humano. O termo "passivo" refere-se ao fato do sensor não gerar e nem irradiar alguma energia para o propósito de detecção.

O *Sensor PIR* possui um circuito integrado que tem por função amplificar os sinais analógicos de detecção e modular, a partir dele, um sinal de saída em nível digital. Assim, quando detectada a presença de um intruso, este sinal de saída permanece em nível alto por um determinado tempo. Caso contrário, ele permanece em nível baixo. Este mesmo sinal vai para a entrada do microcontrolador, que vai interpretá-lo adequadamente. As Figuras 2.10 a) e 2.10 b) ilustram, respectivamente, o

encapsulamento do sensor e o circuito mencionado acima, representado pelo chip *BISS0001*.

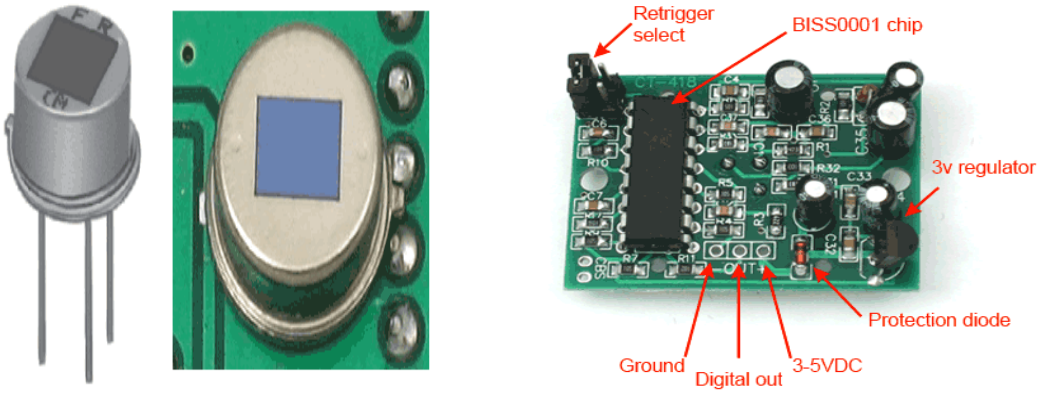


Figura 2.10 a) Encapsulamento b) Circuito integrado

(LADYADA,2013)

Trata-se de um componente de baixo custo e alta eficiência, possuindo um alcance de até 7 metros e um ângulo de abrangência para detecção de movimento superior a 100°. Além disso, por não necessitar de uma alta tensão de alimentação (de 3 a 5V) é ideal para utilização com microcontroladores *PIC* ou outros dispositivos do gênero, que trabalham com tensão de aproximadamente 5V, assim como o *Arduino*.

2.7 Motores de passo

O motor de passo pode ser definido como um transdutor, que converte pulsos elétricos em movimento mecânico de rotação. A rotação do eixo do motor é caracterizada por um ângulo incremental de passo, para cada pulso de excitação. Esse ângulo incremental (que corresponde a um passo) é geralmente especificado na folha de dados do motor e repetido precisamente a cada pulso, que, por sua vez, pode ser gerado por um circuito externo ou por um microcontrolador. O erro que possa existir num determinado ângulo incremental, é geralmente menor que 5%, e não acumulativo.

Motores deste tipo são comumente utilizados quando a aplicação requer uma grande precisão de posição do eixo do motor.O motor de passo possibilita um controle de velocidade, direção e distância, podendo em certas ocasiões, dispensar-se o controle em malha fechada (ou realimentação), bastando para tal que o torque

produzido pelo motor seja suficiente para movimentar a carga acoplada. O circuito responsável pela atuação é constituído por um circuito sequencial (controlador) e um estágio amplificador de saída (*driver*).

De acordo com Jones (1998), motores de passo podem ser classificados em dois tipos, em relação à existência ou não de derivação central nas bobinas que compõem seu enrolamento:

Unipolar : Os motores de passo unipolares são reconhecidos pela derivação central em cada uma das bobinas que o constituem. O número de fases corresponde a duas vezes o número de bobinas, uma vez que cada bobina se encontra dividida em duas. Na Figura 2.11, encontra-se a representação de um motor de passo unipolar de 4 fases (1a, 2a, 1b e 2b).

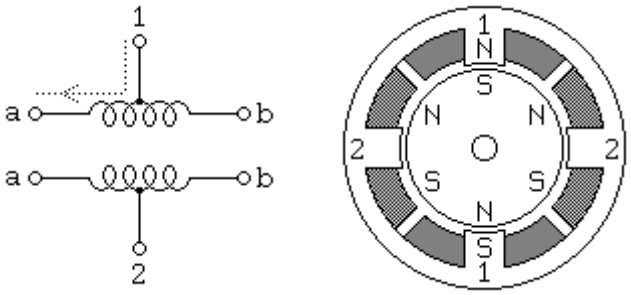


Figura 2.11 - Representação do motor de passo unipolar (JONES,1998)

Normalmente, a derivação central das bobinas é ligada ao positivo da fonte de alimentação e os extremos de cada bobina são ligados seqüencialmente ao terra por um circuito apropriado (controlador mais *driver*), conforme o modo de acionamento adotado, para se produzir um movimento contínuo de rotação.

Bipolar : Os motores bipolares são constituídos por bobinas sem derivação central. Por este fato, estas bobinas devem ser energizadas de tal forma que a corrente elétrica flua na direção inversa a cada dois passos para permitir o movimento contínuo do rotor.Em outras palavras, a polaridade deve ser invertida durante o funcionamento do motor, o que requer um controle apropriado e um pouco mais complexo, geralmente realizado através de um circuito conhecido como *Ponte H*. De forma geral, este circuito eletrônico é constituído por 4 chaves que são acionadas de modo alternado, com a finalidade de permitir que haja a inversão de polaridade.

Na Figura 2.12, representa a configuração das bobinas nos motores bipolares. Nota-se que agora tem-se o número de fases igual ao número de bobinas que compõem o enrolamento do motor.

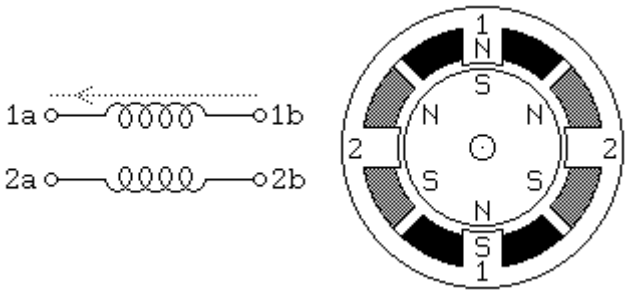


Figura 2.12 - Representação do motor de passo bipolar (JONES,1998)

Os motores de passo bipolares são conhecidos por sua relação tamanho/torque: eles proporcionam um maior torque (aproximadamente 40% a mais), comparativamente a um motor unipolar de mesmas dimensões. Tal questão se deve ao fato de que quando se energiza uma fase, magnetiza-se ambos os pólos em que a fase (ou bobina) está instalada. Assim, o rotor sofre a ação de forças magnéticas de ambos os pólos, ao invés de apenas um, como acontece no motor unipolar.

O funcionamento básico de um motor de passo de ímã permanente, dispositivo de baixo custo altamente utilizado em aplicações não industriais, é condicionado à energização sequencial das bobinas que o constituem, conforme mencionado anteriormente. Na Figura 2.13, observa-se a constituição de motores deste tipo.

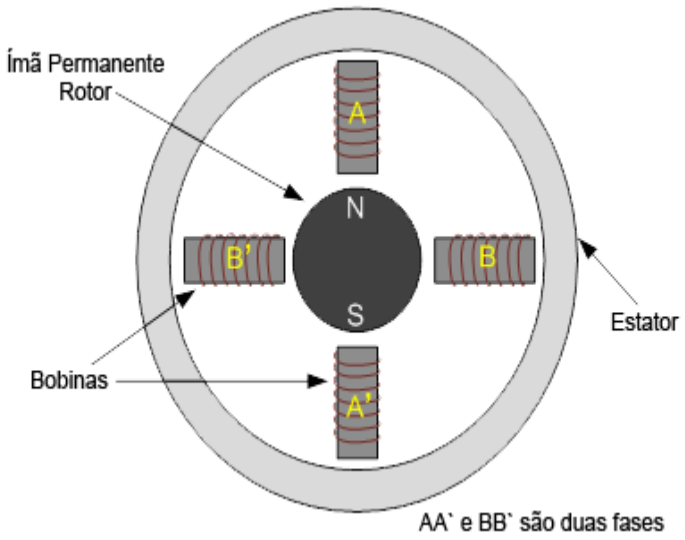


Figura 2.13 - Constituição de motores de imã permanente (KEMPER, 2013)

Ao energizar-se uma bobina, cria-se um campo magnético que atrai o pólo magnético contrário do imã localizado no rotor. Quando se desliga essa primeira bobina e energiza-se a próxima, o imã do rotor segue esse campo magnético e dá um passo no sentido desejado. Há ainda outras alternativas de controle, ao invés de apenas se energizar uma bobina por vez, como por exemplo a técnica de meio passo (*Half Step*). Na Figura 2.14, ilustra-se a sequência de acionamento por meio passo e passo inteiro para motores bipolares e unipolares.

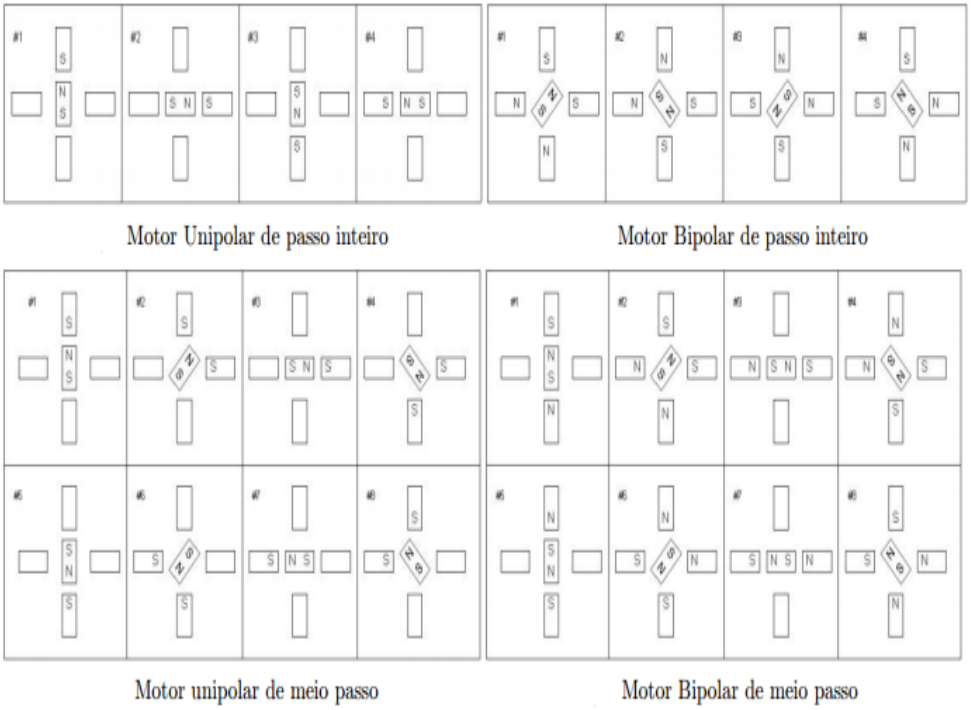


Figura 2.14 - Sequências de acionamento (BRITES, F.G & SANTOS, V.P.A, 2008)

Existem também outros tipos de motores de passo diferentes do motor de imã permanente, como o motor de relutância variável e o motor de passo híbrido, que não serão abordados nesta monografia.

3.Metodologia

Para as aplicações descritas nessa seção, foi inicialmente construída uma página, desenvolvida diretamente na interface de programação do *Arduino,* em linguagem HTML, para ilustrar e implementar a aplicação de um controle de lâmpadas, com botões de controle, de forma a ligar/desligar cada uma delas por meio da intranet e, posteriormente, da internet. O intuito principal da criação desta página foi obter uma maior familiarização com o *Arduino* e compreender as dificuldades envolvidas no processo de execução de comandos e respostas por meio da internet. Entretanto, este não é o objetivo principal desta monografia, e, após validadas as questões já citadas, foi desenvolvido um aplicativo para *Android* objetivando não somente o controle de lâmpadas, já implementado anteriormente, como também o controle de outros dois processos de uma residência: alarme e alimentação de animais de estimação, de acordo com a necessidade do usuário.

O conceito principal de funcionalidade do sistema, de forma geral, é simples: cada botão de controle, inserido no aplicativo, controla, de qualquer lugar do mundo, uma ou mais saídas digitais do *Arduino* que, com auxílio de circuitos auxiliares, realizam a ação desejada. Além disso, o *Arduino* envia *feedbacks* ao usuário, informando qual o estado atual de cada aplicação.

Nesta seção, serão aprofundados estes conceitos, descrevendo e detalhando os métodos utilizados para realização das aplicações mencionadas, bem como a forma de configuração do *no-ip* e roteador para o acesso externo, além da criação do aplicativo com auxílio do *App Inventor.*

O software desenvolvido no *Arduino* e o projeto no *App Inventor*, encontram-se nos Apêndices A e B, respectivamente.

3.1 Configuração para acesso via internet

Para o acesso ao *Arduino* via internet com auxílio do *no-ip* é necessária a realização de alguns passos que serão descritos a seguir. Primeiramente,é importante frisar que a porta recomendada pelo *no-ip* é a porta 80, a padrão de comunicação. Contudo, alguns provedores de internet bloqueiam esta porta, justamente para que não se possa armazenar *sites* em casa. Assim, caso esta não esteja disponível, é necessário redirecionar o fluxo de informações para uma outra porta, que esteja

disponível pelo provedor de internet. Para descobrir se uma porta está disponível ou não, pode-se acessar o *site* (CONYOUSEEME,2013) e testar a porta no espaço correspondente. O serviço tentará então "enxergar" o seu IP público através da porta escolhida. Caso consiga, a porta está aberta pelo seu provedor de internet, caso contrário será necessário tentar uma outra porta, até que se obtenha sucesso. Após descoberta uma porta que esteja acessível, executa-se o processo exemplificado na Figura 3.1, onde o fluxo foi redirecionado para porta 8888 através da configuração no *site* do *no-ip*.

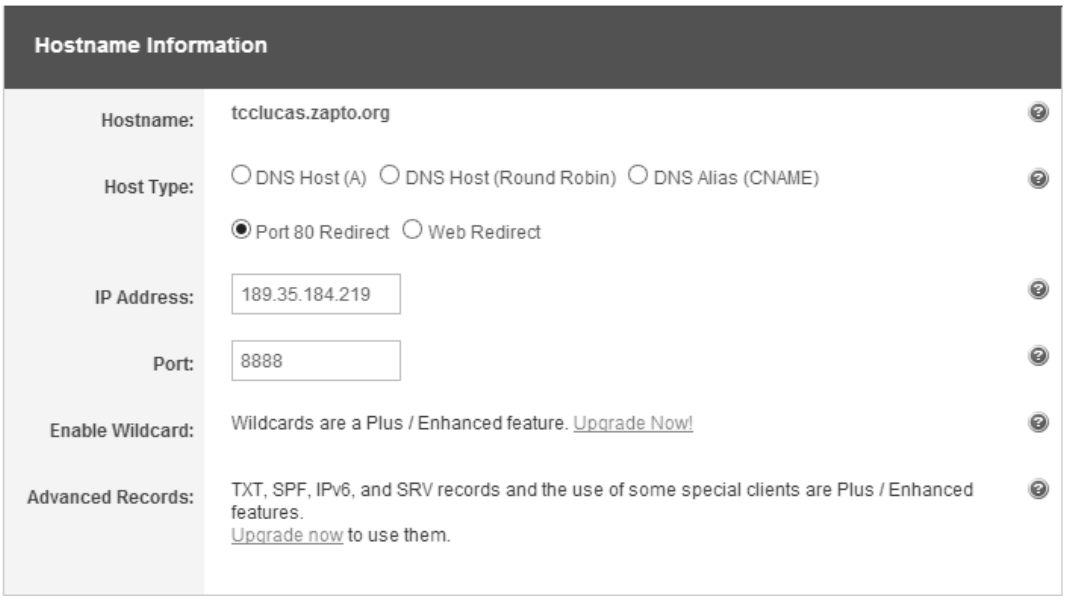


Figura 3.1 - Redirecionamento de fluxo pelo *no-ip*

O "*IP Address*" é o IP atual da rede doméstica, descoberto automaticamente pelo próprio *site*.

Após a configuração do servidor como desejado, é necessário criar manualmente uma regra de roteamento nas configurações do roteador,através da opção "Virtual Server". Toda informação que vier através da porta 8888, é redirecionada para o endereço de IP que foi configurado para o *Arduino* via software. Na Figura 3.2 pode-se observar o ocorrido. O roteador utilizado foi o D-Link-524.

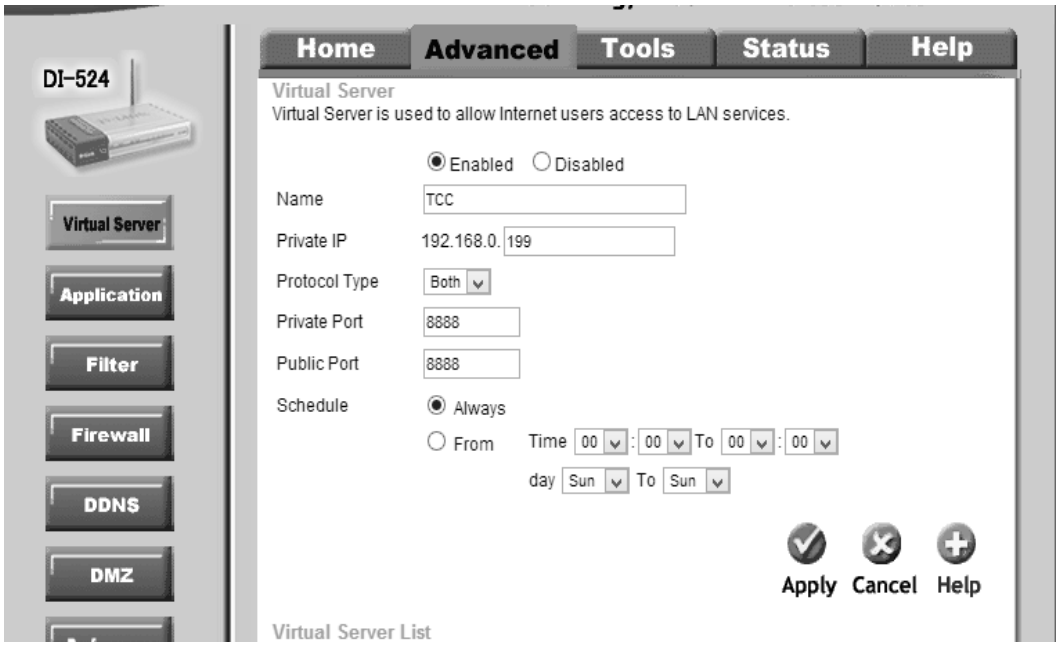


Figura 3.2 - Regra de roteamento

O endereço de IP *192.168.0.199* foi configurado no *Arduino* via software com auxílio da biblioteca do *Ethernet Shield*, como pode ser observado no Apêndice A.

Por último, é necessário configurar o roteador como na Figura 3.3, para atualizar o *no-ip* com o endereço de IP da rede onde se encontra o *Arduino*, evitando que o acesso a ele seja bloqueado mesmo que este IP se altere.

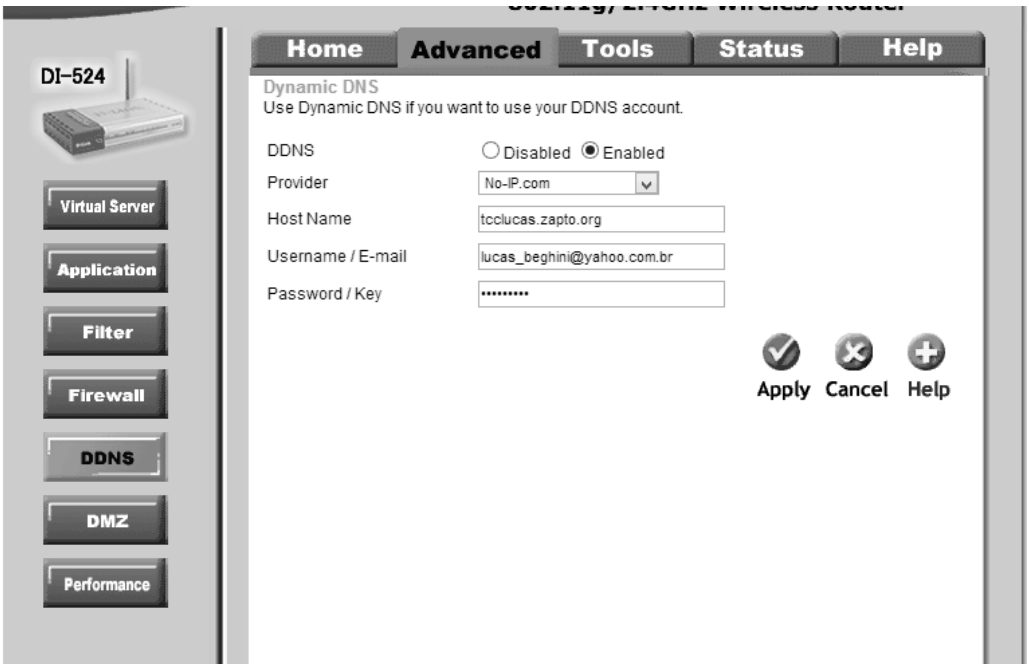


Figura 3.3 - Atualização do DDNS

Desta maneira, o roteador atualiza o *no-ip* periodicamente, informando qual o IP necessário para acessá-lo.

3.2 Controle de lâmpadas

Para o controle de lâmpadas utilizou-se o circuito da Figura 3.4.

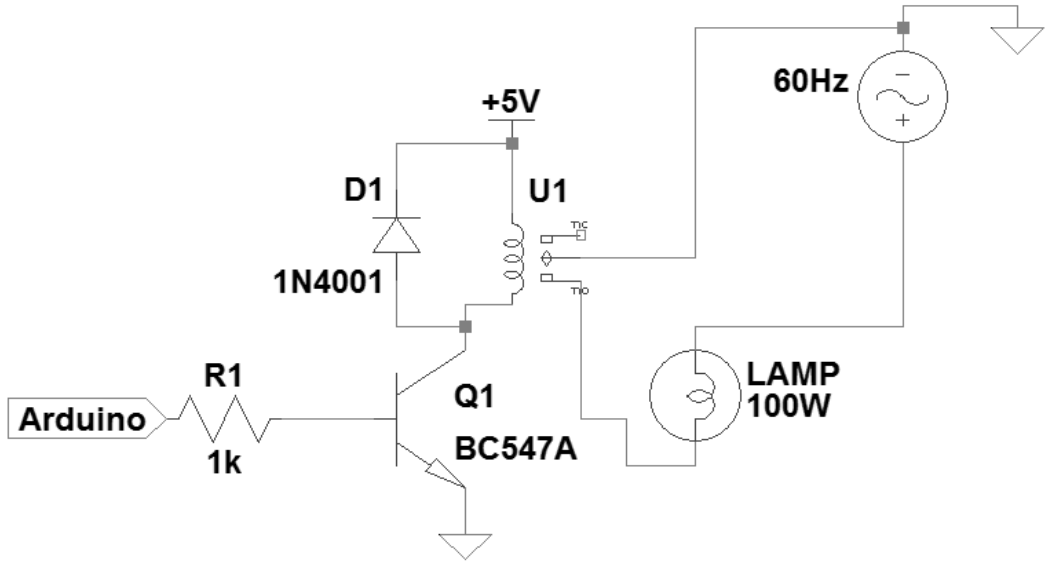


Figura 3.4 - Circuito para controle de lâmpadas

Na base de Q1 encontra-se uma saída digital do *Arduino*, acionada por um botão via internet. Caso haja corrente na base do transistor (saída do *Arduino* em nível alto), este funciona como uma chave, ativando a bobina do relê U1, fechando o contato auxiliar e, consequentemente, acendendo a lâmpada. Caso contrário, ela permanece apagada. O diodo D1 protege o sistema contra força-eletromotriz reversa.

O circuito da Figura 3.4 é bastante comum em aplicações envolvendo microcontroladores. Ele permite que se controle dispositivos que demandem uma corrente da ordem de 100 a 500 vezes maior (dependendo do ganho do transistor utilizado) do que a corrente máxima de saída de uma porta digital do microcontrolador. Para o caso do *Arduino*, por exemplo, e do BC547A, utilizado nesta monografia, é possível controlar uma corrente de até aproximadamente 4A no coletor do transistor, visto que cada porta digital demanda no máximo 40mA de corrente e o transistor citado possui um ganho de corrente próximo a 100. Este fator aumenta muito a quantidade e a variedade de aplicações que o sistema pode suportar. Além disso,o

uso deste circuito constitui também um importante fator de segurança, que impede um possível dano ao sistema. Como a tensão instantânea em uma bobina qualquer é dada pela indutância da mesma multiplicada pela variação da corrente em função do tempo, e como a corrente varia muito rapidamente no indutor, esta tensão instantânea pode assumir um valor muito grande, o que pode ocasionar os chamados surtos de tensão, que por sua vez podem queimar o transistor ou afetar o microcontrolador. Por isso, a utilização do diodo D1, que impede que estes surtos cheguem ao circuito de aplicação, mantendo sempre uma tensão de no máximo 0,7V (queda de tensão no diodo) a mais que a fonte de alimentação na bobina, valor que o transistor suporta sem maiores problemas.

Na prática, é recomendável a utilização do relê substituindo um interruptor paralelo de forma que, ou um, ou outro, liguem ou desliguem o sistema quando for desejado. Assim, é possível controlar as lâmpadas pela internet ou da forma convencional. O esquema de ligação proposto encontra-se na Figura 3.5.

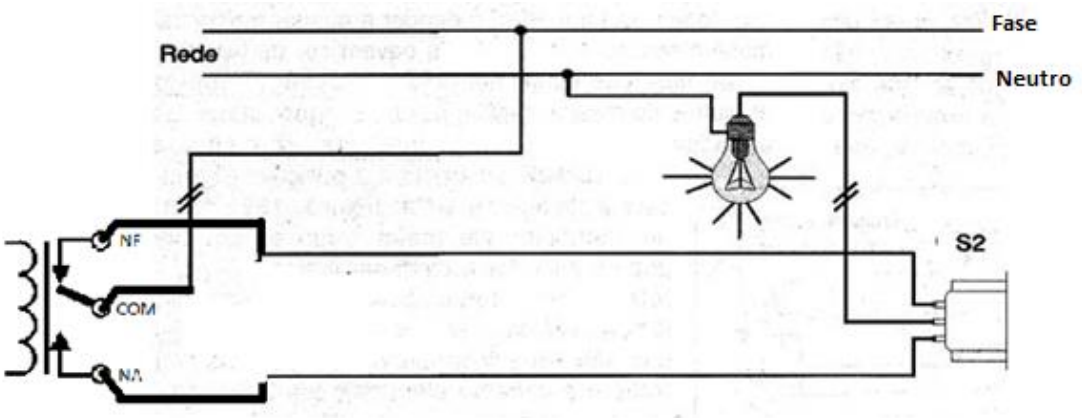


Figura 3.5 - Interruptor paralelo e relê

Para este trabalho, foi construído um circuito para apenas uma lâmpada independente. Contudo, o conceito é extensível a várias lâmpadas, caso este seja o intuito do projeto.

3.3 Alarme residencial

O circuito utilizado para implementação do alarme pode ser observado na Figura 3.6.

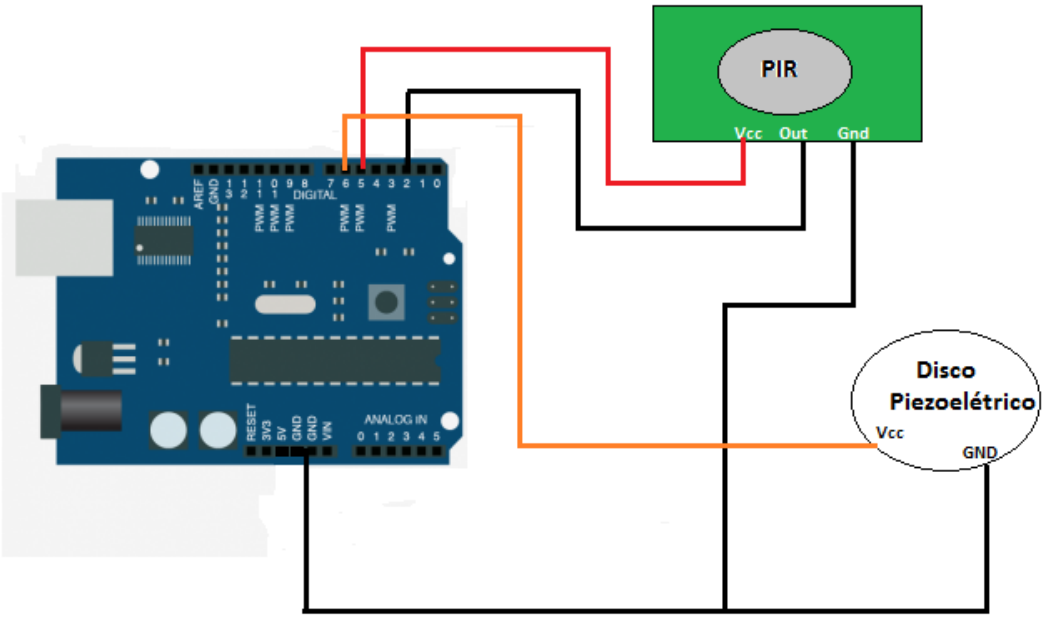


Figura 3.6 - Circuito de implementação do alarme

Para simulação de uma sirene utilizou-se um disco piezoelétrico. Trata-se de um dispositivo simples, feito de uma fina camada de cerâmica, envolta por um disco metálico. Materiais piezoelétricos, tem a capacidade produzir eletricidade quando a eles é aplicada alguma pressão mecânica. O efeito contrário também é verdadeiro. Ao se aplicar um campo elétrico em um material deste tipo, o mesmo muda de forma, provocando um efeito sonoro audível conforme o disco se deforma. Portanto, ao ser aplicada uma tensão nos terminais de um objeto deste tipo,por meio de uma das saídas digitais do *Arduino*, o disco passará a vibrar em uma única frequência, produzindo sons.

Após ter sido desenvolvido o circuito de aplicação e a parte de software para comandar a leitura do sensor de movimento e ativação da sirene quando for detectada a presença de intrusos, implementou-se as notificações ao usuário quando o alarme é disparado. A priori, a intenção era utilizar as próprias notificações do aplicativo, da mesma forma que outros aplicativos conhecidos utilizam, como o *Facebook* e o

Whatsapp, por exemplo. Porém, embora seja uma requisição constante dos usuários nos fóruns especializados, o *App Inventor* ainda não possui uma forma de gerar notificações em tempo real. Ele não tem a capacidade de manter as aplicações "vivas" enquanto o celular não está sendo utilizado. Dessa forma, caso o usuário não esteja utilizando o aparelho móvel, ele não verificaria o disparo do alarme em tempo real, mas somente quando voltasse a utilizá-lo. Partiu-se então para uma outra solução que resolvesse este impasse e o método escolhido foi a utilização do *Twitter*.

O *Twitter* é uma rede social e servidor para *microblogging*, que permite aos usuários o envio e recebimento das atualizações pessoais de outros contatos (em textos de até 140 caracteres, conhecidos como "tweets"), por meio do *website* do serviço, por SMS e por softwares específicos de gerenciamento. O serviço é gratuito pela internet e qualquer usuário com acesso a rede ou a algum dispositivo móvel com essa funcionalidade pode criar uma conta e usufruir do sistema sem ônus financeiro. Estima-se que atualmente cerca de 500 milhões de pessoas ao redor do mundo utilizem a rede social.

No segundo semestre deste ano de 2013 o *Twitter* atualizou seu sistema para aplicações de celular implementando as "Notificações Push" (em tempo real), assim como outros aplicativos já utilizavam. Ademais, ele possui a interessante característica de poder enviar SMS do servidor diretamente para celulares também de forma gratuita (por enquanto somente para as operadoras TIM e Nextel, pois as outras só permitem o envio do SMS do celular para o *Twitter*, e não ao contrário), em situações específicas que o usuário pode configurar dentro do próprio *microblog* (como menções ou tweet de algum amigo específico). Particularmente, esta é uma característica importante para a aplicação desejada, pois elimina a necessidade de estar em um local com acesso a internet para receber a notificação proveniente do alarme, visto que ela pode ser feita via mensagem *SMS*.

Embora seja uma ferramenta interessante, o acesso ao *Twitter* por meio do *Arduino* não é um processo fácil e foi mudado recentemente. Segundo McRoberts (2011,p.420),

> A partir de 31 de agosto de 2010, o Twitter alterou sua política no que se refere ao acesso ao site utilizando aplicativos de terceiros. Um método de autenticação, conhecido como *OAuth*, agora é utilizado, tornando muito mais difícil "twittar" diretamente a partir de um *Arduino*; antes dessa alteração, esse processo era muito mais fácil. Enviar mensagens pelo Twitter, neste momento, só é possível

> utilizando um recurso externo. Em outras palavras, é necessário enviar o tweet para um site, ou proxy, que twittará em seu nome, utilizando o token *OAuth* (código de acesso).

O protocolo *OAuth* fornece uma forma padronizada de acessar dados protegidos. De fato, a biblioteca do *Twitter* e o código exemplo, que vem inseridos com a *IDE* (interface de programação) ao baixá-la, não funcionam mais por esse motivo. Assim, utilizou-se um servidor *proxy* para tornar possível as postagens por meio do *Twitter*.

Servidor *proxy* é um servidor que atende a requisições repassando os dados do cliente a frente. Um usuário conecta-se a ele, requisitando algum serviço (neste caso o acesso ao servidor do *Twitter*) e este repassa a requisição para o servidor destinatário. Um servidor proxy pode, opcionalmente, alterar a requisição do cliente ou a resposta do servidor e, algumas vezes, pode disponibilizar este recurso sem nem mesmo se conectar ao servidor especificado. Em redes de computadores, pode também atuar como um servidor que armazena dados em forma de cache, guardando informações para melhorar a rapidez de conexão a servidores que já foram acessados anteriormente.

O servidor citado como *proxy* é o (ARDUINOTWEET,2013). O *site* foi desenvolvido por um usuário do *Arduino*, denominado *Neocat,* e que utiliza seu domínio para enviar o tweet com o protocolo de autenticação *OAuth.* Para enviar um "tweet" é necessário instalar a biblioteca *Twitter.h*, desenvolvida por ele, substituir a anterior e obter um *token* (versão criptografada do seu nome de usuário e senha), ambos disponíveis em seu site. Ainda há um exemplo de código a se implementar para "twittar". Embora dezenas de usuários utilizem este mesmo caminho para "twittar", recomenda-se, por questões de segurança ao utilizar um servidor *proxy*, a criação de um novo perfil anônimo no *Twitter* apenas com o intuito de monitoramento de sua residência, assim como foi feito neste trabalho.

Realizada a conexão, configurou-se dentro do próprio *Twitter* a obtenção de notificações toda vez que o perfil *casamonitor*, criado exclusivamente para este projeto, mencionar o perfil pessoal do autor desta monografia. Assim, basta configurar, via software, na *IDE* do *Arduino*, a mensagem a ser "twittada" como uma menção ao perfil que se deseja notificar. A forma de fazer isso através do *Twitter* é "@ + nome do usuário".

Finalmente, é necessário baixar o aplicativo do *Twitter* em seu celular *Android*, cadastrar sua conta pessoal no mesmo e ativar a sincronização de conta para permitir também as notificações em tempo real.

3.4 - Alimentação de animais de estimação

Para viabilizar o desenvolvimento do sistema de alimentação via internet, simulou-se o uso do motor de passo unipolar *Mitsumi M42SP-5* (ou similar),controlado pelo *Arduino*, que é responsável por abrir e fechar o compartimento onde fica armazenada a comida do animal, despejando a mesma no recipiente adequado, a partir de um comando dado pelo usuário ao aplicativo instalado em seu celular. Um outro comando é dado para fechar o compartimento e interromper a alimentação. Além disso, utilizou-se uma *Câmera IP* para se obter imagens do recipiente de comida, indicando se há ou não algum alimento no mesmo. O fluxograma é representado na Figura 3.7.

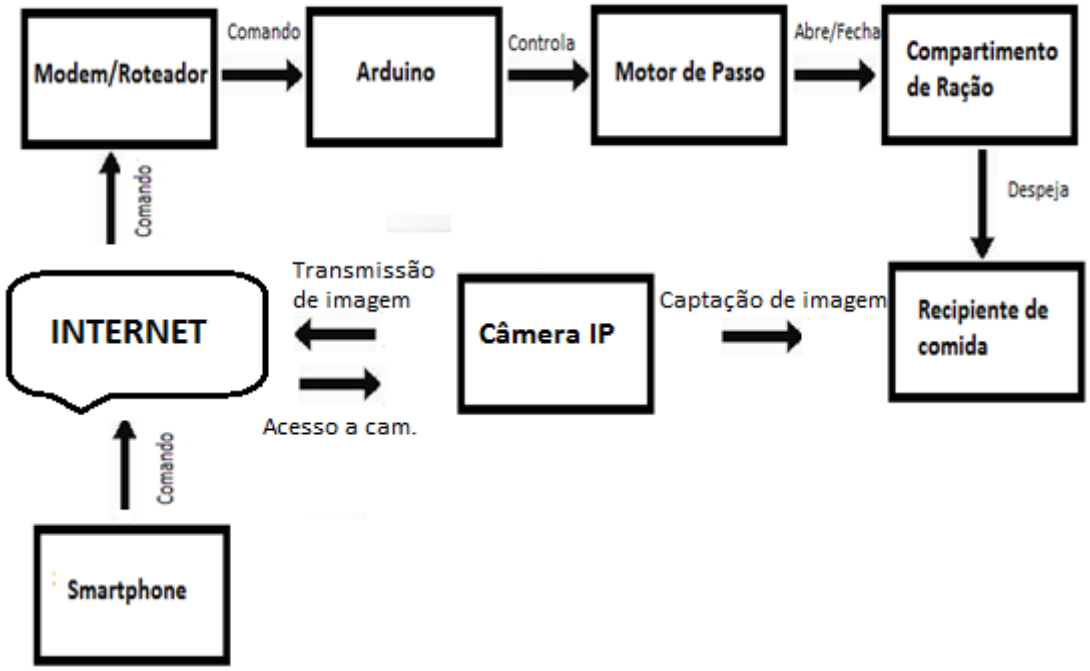


Figura 3.7 - Fluxograma para alimentação via internet

Uma *Câmera IP* é um dispositivo capaz de transmitir imagens e receber informações através da internet. Ela é conectada ao roteador via *Wireless* (existem também modelos que são conectadas por cabo de rede) e portanto também possui um

IP na rede. Conforme já foi discutido na Seção 2.4, para acessá-la externamente é necessário a utilização de um servidor DDNS, que pode ser disponibilizado pelo *no-ip*. Dessa forma, utilizou-se para acesso a câmera o mesmo procedimento descrito na Seção 3.1, criando-se um novo *Host* denominado (*cameracanil.zapto.org)* e utilizando-se, desta vez, a porta 8080 de comunicação. Portanto é possível acessar as imagens do local onde a câmera se encontra de qualquer dispositivo que tenha conexão com a internet, como por exemplo um *smartphone*, bastando acessar o domínio descrito acima através do navegador. O modelo de câmera utilizado foi a *Edimax IC-3110W*, semelhante à ilustrada na Figura 3.8.

Figura 3.8 - Câmera IP (STARDOT,2013)

Para simulação do motor de passo, foi selecionado um motor com configuração unipolar, que possui um controle mais fácil de ser implementado, por não necessitar de inversão de polaridade das bobinas, conforme explicado na Seção 2.7. Foram ligados 4 *Leds*, cada um a uma saída digital do *Arduino,* simulando as 4 bobinas de um motor de passo, que, energizadas sequencialmente, são responsáveis pelo movimento do mesmo. Quando for necessária a troca de sentido, basta inverter a sequência. Entretanto, cabe ressaltar que na prática não é possível ligar os 4 fios provenientes das bobinas do motor direto na saída do microprocessador por questões já apontadas anteriormente. É necessário um circuito de acionamento semelhante ao da Figura 3.4 para cada bobina ou senão um driver que o substitua, como por exemplo o *CI ULN2003*. A Figura 3.9 mostra a ligação do motor de passo *Mitsumi M42SP-5* juntamente com o *ULN2003*. Também é necessária uma fonte de alimentação externa para alimentar o motor de 12 a 24VDC.

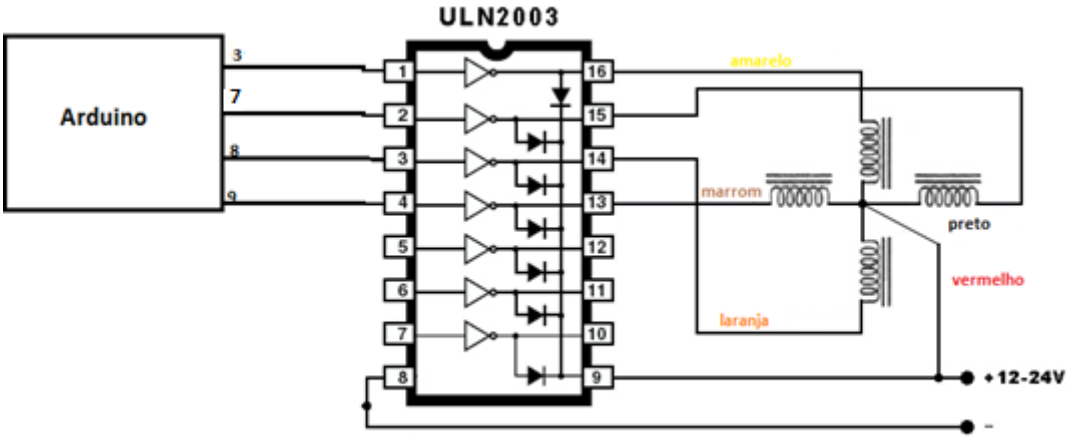


Figura 3.9 - Esquema de ligação motor de passo

Já a Figura 3.10, representa um desenho ilustrativo (fora de escala) do sistema de alimentação, feito a mão livre e digitalizado posteriormente, onde a central de automação corresponde ao *Arduino* e o *Ethernet Shield*.

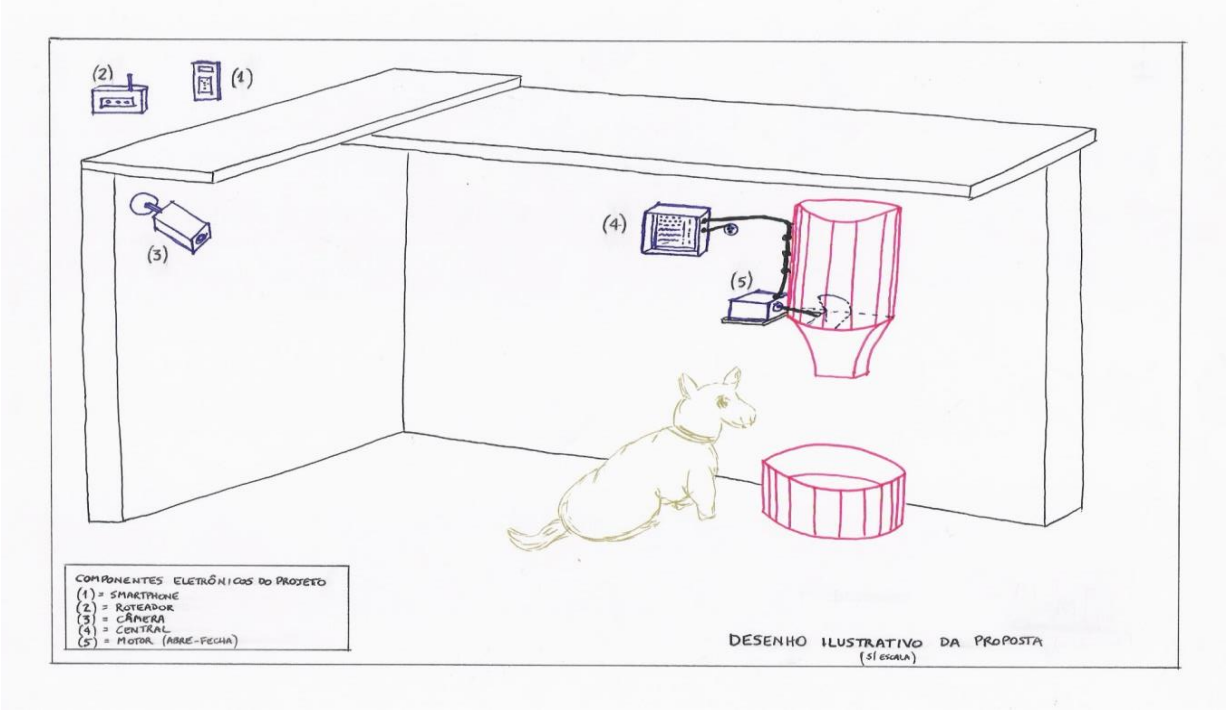


Figura 3.10 - Desenho ilustrativo do sistema de alimentação

3.5 Criação do aplicativo

Por fim, desenvolveu-se um aplicativo capaz de comandar todas as aplicações descritas anteriormente. Utilizou-se neste processo o *App Inventor*, detalhado na Seção 2.5. As 3 telas de interação com o usuário e os respectivos componentes utilizados para posterior tratamento no Editor de Blocos, estão representados na Figura 3.11

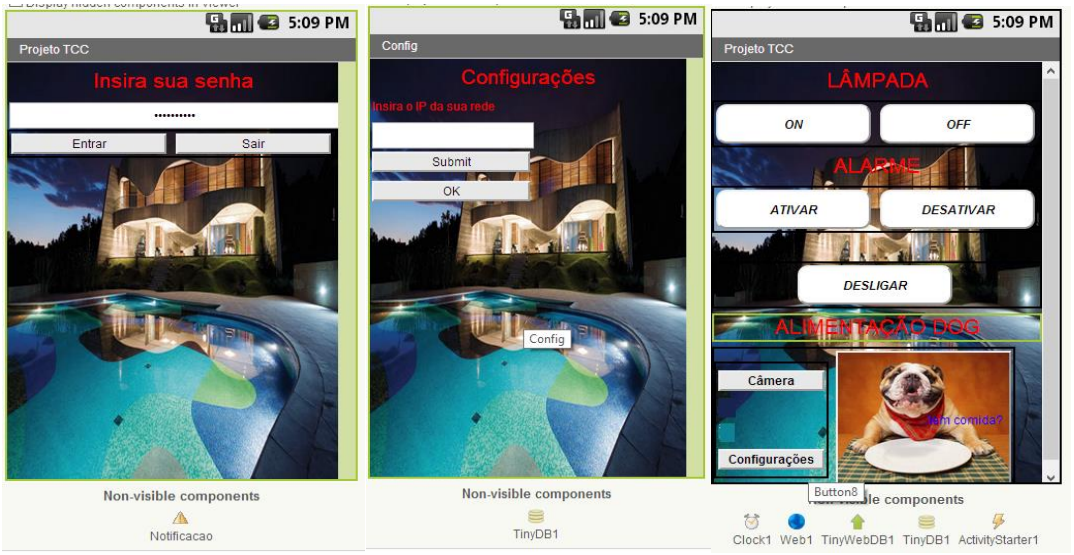


Figura 3.11 - Telas de interação com o usuário

A primeira tela é a tela inicial, onde será necessário inserir a senha do usuário para permitir o acesso às outras telas. Dessa forma, em caso de furto do celular ou em outra circunstância onde o smartphone possa ficar exposto a outros indivíduos, ninguém estará apto a acessar a tela principal de controle da residência, a menos que possua a senha do aplicativo. A senha para acesso é definida no Editor de Blocos, e para o caso desta monografia foi definida como "tcc". Em caso de senha incorreta, exibi-se uma notificação alertando para este fato. O botão "Sair" fecha a aplicação.

A tela "Config" é a tela onde será necessário incluir o IP público da rede que hospeda o *Arduino*. Inicialmente, tentou-se fazer esse acesso através dos métodos já citados nesta monografia, com auxílio do *no-ip.* Alguns problemas, que serão discutidos no próximo o capítulo, ocorreram e foi necessário o desenvolvimento de um outro método para substituir o serviço de DDNS fornecido pelo *no-ip*.

Para solução deste impasse, foi elaborada uma solução através do próprio *Arduino.* A nova função designada a este consiste em acessar um *site* especializado

em descobrir IP's, periodicamente, interpretar esse dado fornecido pelo *site* e verificar se o mesmo é igual ou diferente do dado armazenado na memória. Se for diferente, significa que o IP da rede mudou, e então novamente utiliza-se o *Twitter* para notificar o usuário que o IP foi modificado e qual é esse novo número. O usuário então, obtendo essa informação, acessa a tela de configurações do aplicativo a atualiza o IP da sua rede para esse novo número, inserindo-o e clicando em "Submit".

O *site* escolhido foi o (CHECKIP,2013). Ao acessá-lo, obtêm-se o seguinte cabeçalho no formato HTML:

<html><head><title>CurrentIP Check</title></head><body>Current IP Address: 177.34.170.35</body></html>

que pode ser decifrado de modo a se obter somente o endereço numérico do IP. O trecho de código correspondente a esta função foi adaptado do *blog* (JOSEMATM,2013).

Interpretado o IP da rede, esse dado é armazenado na memória de programa do microcontrolador do *Arduino*, verificando, a cada acesso ao *site*, se o novo valor obtido é igual ou diferente do armazenado. Em caso de diferença, esse novo valor será armazenado na memória de programa, mas será ainda necessária uma conversão de tipos de dados para transmiti-lo ao Twitter. O número de IP devolvido pela função é um numéro de 32 bits ou do tipo *uint_32 .* Portanto,será necessário transformá-lo em 4 vetores de 8 bits (ou *uint_8)* e depois imprimí-lo em um vetor de caracteres (através da função *sprintf*) antes de transmití-lo com auxílio da biblioteca *Twitter.h.*

Realizada a conversão de dados e a transmissão, o usuário é notificado e cabe a ele acessar o aplicativo e atualizar esta informação. Não é necessário colocar este número todas as vezes que abrir o aplicativo. Este número de IP fica armazenado no celular, a menos que se exclua os dados do aplicativo nas configurações do aparelho ou que se instale o mesmo novamente. Uma vez colocado o dado e em caso de não se ter recebido notificações, basta teclar "ok" quando acessar a tela "Config" que o usuário será direcionado à tela principal.

Na tela principal, encontram-se todos os comandos para as 3 aplicações desenvolvidas. Os botões interpretam se as saídas do *Arduino* estão em nível alto ou baixo, indicando ao usuário, por meio de cores (verde para ligado e vermelho para desligado), qual o status de determinada aplicação. O botão "Desligar" é não somente o botão que desliga a sirene do alarme, mas também um indicador de status se o

alarme está disparado ou se está desligado, mudando de cor e indicando visualmente ambos os casos. Por fim, para a alimentação canina, é possível acessar a câmera localizada no canil e visualizar se há comida ou não no recipiente ou também se não há nada de errado com o animal. Caso o usuário necessitar alimentar o cão, basta o mesmo clicar na imagem indicada. Se o compartimento estiver fechado, o texto da imagem permanecerá pedindo por comida. Em contrapartida, se o mesmo estiver aberto, o cão exibe uma mensagem de agradecimento. O botão de configurações retorna para a tela "Config" caso seja necessária alguma alteração na mesma.

Segue abaixo uma descrição de todos os componentes do *App Inventor* utilizados e suas funções dentro do sistema:

Notifier: Os blocos que estão presentes nesse componente são capazes de exibir vários tipos de notificações ao usuário do aplicativo, incluindo mensagens SMS ou simples alertas na forma de texto.Foi utilizado neste projeto com a finalidade de notificar o usuário na tela inicial, no caso deste ter inserido uma senha incorreta para tentar acessar o aplicativo.

Tiny DB: Permite que o usuário armazene informações dentro do próprio aplicativo e as recupere a qualquer momento. Este recurso foi utilizado para armazenar a IP pública do usuário na rede e repassá-la à variável "ip", utilizada na Tela Principal. Para utilização deste componente em telas diferentes, como aconteceu neste estudo, foi necessário incluí-lo em ambas, conforme observado na Figura 3.11.

Clock: Fornece ao usuário a opção de criar eventos que executem determinada ação em intervalos regulares de tempo definidos pelo usuário. Para este trabalho, sua função foi prover um intervalo de tempo de 1s para se acessar o domínio onde se encontra o *Arduino* (com auxílio do componente *Web*), de forma a permitir a obtenção da resposta das portas digitais, fornecendo o *feedback* para o aplicativo periodicamente. Esse componente só funciona quando o celular está ativo, e esta é justamente a questão determinante que inviabilizou as notificações em tempo real pelo *App Inventor*.

Web: Componente que fornece funções para requisição HTTP GET e HTTP POST. Esse foi o componente utilizado para se fazer uma requisição e acessar o domínio gerado pelo *Arduino* a partir da variável "ip".

TinyWebDB: Componente que se comunica com um domínio na internet e tem por função transmitir, armazenar e recuperar informações. Foi utilizado para passar as intruções ao *Arduino*, via internet com auxílio do componente *Web*.

Activity Starter: Contém funções que conseguem iniciar atividades dentro do aplicativo, como abertura de novos ícones, acionamento da câmera do aparelho, dentre outras. Foi utilizado como um botão com o link necessário para se acessar a *Câmera IP* a partir do navegador por meio do endereço que ela possui na internet, gerado pelo *no-ip*.

O projeto para cada uma das telas, com as funções fornecidas pelos componentes citados, encontra-se no Apêndice B.

4. Resultados e Discussões

Uma amostra da página HTML construída para testes de comandos ao *Arduino* através da internet, pode ser visualizada na Figura 4.1.

Figura 4.1 - Amostra da página

O acesso ao *Arduino* através dos botões da página HTML foi realizado com sucesso, sendo testado na Intranet e também na Internet, esta última forma de acesso realizada com auxílio do *no-ip*, através do domínio (*tcclucas.zapto.org)*, após todo o procedimento descrito na seção 3.1. Implementou-se também o circuito da Figura 3.4 para testes do acionamento de lâmpadas via internet, também realizado com sucesso. A respeito da confecção da página, esta questão não foi aprofundada por não ser o objetivo deste trabalho. Uma vez testada a conexão com o *Arduino* via internet para o acionamento de lâmpadas, foi dado prosseguimento ao trabalho com a criação do aplicativo, este sim objetivo principal do mesmo.

A respeito da criação do aplicativo, o principal problema apresentado foi justamente o acesso externo ao *Arduino*. Inicialmente, foi feita a tentativa de realizá-lo através do acesso ao domínio gerado pelo *no-ip*, de maneira análoga à realizada através da página HTML. Dessa maneira, não seria necessário saber qual o IP do *Arduino* na rede, bastava decorar o endereço do domínio, mesmo que este mudasse com decorrer do tempo, conforme foi exposto na Seção 2.4.2. Definiu-se então uma variável "ip" no Editor de Blocos da Tela Principal do *App Inventor* e atribuiu-se a ela o endereço do domínio (com letras do alfabeto). Toda vez que se precisasse fazer uma requisição ao servidor (HTTP GET), essa variável seria chamada. Todavia, não foi

possível realizar a requisição desta forma, através do componente *Web*. O indicativo do app era de que era impossível a obtenção de uma resposta do servidor requisitado, embora o domínio apontasse para o endereço numérico do IP público da rede onde se encontra a central de automação. No entanto, quando a variável "ip" foi colocada diretamente na forma numérica, com o IP da rede, a requisição foi feita com sucesso.Tratou-se portanto de um problema de DNS, onde não foi possível traduzir o endereço com letras do alfabeto para o formato numérico, utilizando os blocos disponíveis no *App Inventor*. Após muitas pesquisas, estudo de todos os componentes do *App Inventor* e contato via internet com fóruns e *blogs* especializados no assunto, não foi possível solucionar essa questão.

Foi implementada uma outra solução, já descrita na Seção 3.5, para substituir o serviço de DDNS fornecido pelo *no-ip*. Inicialmente, essa solução sobrecarregou o sistema, pois agora o *Arduino* teria que acessar o *site* que identifica o IP público da rede a cada execução do código, que fica em um *loop* infinito, o que acarretava em uma requisição a cada 2 segundos, aproximadamente. Dessa forma, o tempo de resposta do sistema para qualquer requisição proveniente do aplicativo, aumentou cerca de alguns segundos para ser processada. A solução para este caso foi inserir uma variável contadora ("cont") no código do programa, de modo que o *site* só fosse acessado após esta atingir um valor determinado. Assim, a requisição agora é feita de cerca de 5 em 5 minutos, evitando a sobrecarga do sistema. Um outro problema encontrado foi que, se em alguma requisição ao site, ocorresse algum problema, ou de origem do provedor de internet ou do próprio servidor que hospeda o *site*, o valor obtido para variável que identifica o IP da rede ("ipAtual") era 0.0.0.0. Como essa variável é diferente daquela que estava armazenada na memória de programa, era postada uma notificação ao usuário, através do *Twitter*, informando que o IP havia mudado e que o novo valor era 0.0.0.0, o que obviamente não se tratava se uma informação verdadeira. Esse problema foi solucionado criando-se uma variável "ipnulo" e atribuindo-se a ela o valor do IP com os zeros, de forma a só enviar a notificação ao usuário se a variável obtida através do *site* fosse, não somente diferente da variável contida na memória de programa, mas também diferente da variável "ipnulo". Ademais, a solução de notificação ao usuário funcionou corretamente, assim como todas as outras funcionalidades do aplicativo. Na Figura 4.2, pode-se observar o funcionamento do mesmo.

Figura 4.2 - Aplicativo em funcionamento

A respeito das aplicações desenvolvidas, cabem algumas considerações. Em relação ao alarme, a solução implementada por meio de notificações via *Twitter* mostrou-se eficaz, apontando todas as vezes que o alarme foi disparado, imediatamente. Foi feita também a tentativa de fazer o sistema de notificações via *email*, através do protocolo *SMTP (Simple Mail Transfer Protocol)*. Trata-se de um protocolo padrão de envios de emails através da internet, relativamente simples, onde um ou vários destinatários são especificados e depois transfere-se a mensagem. A conexão com o servidor de email destinatário foi realizada com sucesso, mas a mensagem não foi transferida. Isso ocorreu devido aos servidores de email usarem atualmente o protocolo de autenticação *TLS (Transport Layer Security)* ou o seu antecessor, o *SSL (Security Sockets Layer)* para receber e transmitir mensagens.

Os protocolos *TLS/SSL* são protocolos baseados em criptografia, que estabelecem segurança de comunicação via internet para alguns serviços específicos, como email *(SMTP)*, por exemplo. São protocolos complexos que, por serem

criptográficos, exigem uma grande capacidade de processamento e, portanto, são de impossível realização em um *Arduino Uno* ou em qualquer outro microcontrolador de 8 bits. Contudo, cabe ressaltar que a solução por meio do *Twitter* atendeu completamente às solicitações e que é também possível configurar, dentro da própria rede social, o envio de *emails* toda vez que houver uma menção de perfil, se o usuário julgar necessário este recurso, além das notificações em tempo real através do aplicativo.

Tratando-se ainda da questão das notificações, o *Twitter* também é capaz de enviar mensagens *SMS* gratuitamente para o número de celular especificado nas configurações da rede social, conforme mencionado na Seção 3.3. Dentro desse contexto, o *Arduino* também possui um *shield* específico para enviar *SMS* para celulares.Trata-se do *Shield GSM*. Para o envio de mensagens, é necessário o uso de um chip para celulares, o que acarreta custos adicionais de acordo com a operadora escolhida. Entretanto, o uso deste *shield* é uma alternativa interessante, pois pode funcionar como redundância ao sistema, em caso de alguma falha proveniente do *Twitter* ou do servidor *proxy* que faz a requisição a este serviço para o envio das notificações,ou para simples envio de *SMS*, para usuários que não possuem celulares das operadoras *TIM* ou *Nextel* (as únicas que suportam o serviço de mensagens *SMS* através do *Twitter* neste momento).Já para os usuários que não possuem acesso a internet pelo seu smartphone em tempo integral, este *shield* é de fundamental importância, pois as notificações não dependerão mais do acesso à rede.

Finalizando a discussão a respeito do alarme residencial, cabe ressaltar que o disco piezoelétrico para produção de sons foi escolhido apenas para efeito de simulação, não dispondo de uma capacidade sonora suficiente para ser aplicado a um alarme residencial. Para um melhor desempenho deste tipo de dispositivo recomenda-se o uso de uma outra sirene mais potente, sendo necessário o uso de circuitos auxiliares para sua ativação, semelhantes ao da Figura 3.4, utilizado para controle de lâmpadas.

Em relação à alimentação de animais de estimação, também é importante fazer algumas considerações. A escolha da *Câmera IP*, deveu-se, sobretudo, a substituição de todos os sensores que seriam necessários para monitoramento da aplicação. Graças a sua versatilidade, é possível monitorar não somente o animal, mas também se há comida no recipiente de ração e se há comida no compartimento de alimentação, substituindo eventuais sensores que teriam que ser utilizados para essas finalidades.O modelo de câmera descrito neste trabalho não dispõe de movimento

mecânico e rotativo controlado via internet, mas existem determinados modelos que possuem essa funcionalidade, essencial para monitoramento dessa aplicação. Ademais, é também necessária uma atenção especial ao local onde a câmera está instalada, pois ela não possui uma resolução adequada à longa distância, impossibilitando uma visão nítida do sistema como um todo nessa ocasião. Se não for possível instalá-la em local adequado para suprir essa necessidade, será necessário fazer um tratamento de imagem antes da mesma chegar a seu destino final, que é o celular. Mais uma vez, esbarra-se nas limitações de capacidade do *Arduino Uno* para realizar esta tarefa, impossibilitando o usuário de realizá-la através deste dispositivo.

Também foi considerada a possibilidade de fazer um monitoramento do recipiente de comida por meio de um sensor de força resistivo, que identificaria, a partir da massa total do recipiente, se seria necessário, ou não, a alimentação do animal. Particularmente, seria um artifício proveitoso, pois forneceria a possibilidade de notificar o usuário na falta de alimento. Contudo, por necessitar de testes com um protótipo construído, este sistema não foi implementado. Ainda assim, seria necessária sua atuação em conjunto com a câmera, para comprovar que a notificação não estava sendo mascarada por um peso extra no recipiente, que pode ser inclusive o próprio corpo do animal.

Por fim, é importante destacar o papel do *Arduino* no sistema como um componente totalmente passivo em relação ao aplicativo que o comanda. Embora o mesmo identifique se as portas do *Arduino* estão em nível alto ou baixo, informando ao usuário qual o estado atual das aplicações, a requisição é sempre feita a partir do celular e nunca da central de automação. Para o *Arduino* fazer uma requisição ao aparelho, é necessário um servidor intermediário, como no caso deste trabalho, o *Twitter*. Isso se deve ao fato da inexistência de um servidor presente no celular, que possibilitaria a comunicação direta entre ambos. Já existem alguns servidores para *Android* desenvolvidos, e futuramente podem ser tornar alternativas interessantes na questão da automação residencial.

5.Conclusões

A partir dos fatos relatados e resultados expostos, conclui-se que foi possível realizar o objetivo deste projeto através dos métodos propostos. No contexto atual do controle de residências através de sistemas embarcados, utilizando microprocessadores como central de automação, o *Arduino Uno* mostrou ser uma ferramenta de fácil implementação e com uma boa relação custo-benefício para o controle de alguns processos residenciais, embora limitado em alguns aspectos, como capacidade de processamento de dados e portas de saída disponíveis. Além disso, a utilização do *App Inventor* para criação de aplicativos mostrou ser uma alternativa viável para aproximar os usuários do sistema operacional *Android*, que não tem experiência em programação *Java*, com a criação de aplicativos, área cada vez mais explorada com a popularização de *smartphones* e *tablets*.

Ademais, a área ligada a automação residencial está em crescente evolução, e a tendência é a utilização de sistemas mais robustos e com maior capacidade de processamento de dados, integrando o maior número possível de aplicações e fazendo com que o binômio custo-benefício tenha cada vez mais importância.

Em linhas gerais pode-se concluir que os benefícios gerados pelo aplicativo com baixo custo de investimentos apontam um cenário de possibilidades que pode ser extendido a outros segmentos, usando sistemas similares, como por exemplo o setor de saúde: monitoramento de pacientes, de crianças e idosos, adequando-os às necessidades de cada usuário.

5.1 Trabalhos Futuros

Para trabalhos futuros, sugere-se a utilização de uma ferramenta com mais recursos que o *Arduino Uno*, como por exemplo o *Arduino* Mega, que possui mais capacidade para processamento de dados e portas de saída, a fim de que se possa controlar mais processos residenciais. Também para aplicações consideradas de grande porte, recomenda-se o uso do *Wi-fi shield*, onde não é necessário a conexão diretamente com o cabo de rede, já que a mesma é feita via *Wireless*. A conexão via cabo de rede pode inviabilizar aplicações onde o *Arduino* necessita ficar em uma distância muito grande do roteador.

Contudo, também merecem ser analisadas outras plataformas de hardware e software atuando em sistemas embarcados, como por exemplo a *RaspBerryPi*, principalmente atuando em conjunto com o *Arduino* na central de automação. Entretanto, essa plataforma, embora seja uma ferramenta muito mais poderosa, exige do desenvolvedor um pouco mais de experiência em programação e um conhecimento de *Linux.*

Pode-se também construir o sistema de alimentação canina, verificando a questão do sensoriamento por sensores de força resistivos e implementando notificações ao usuário na ausência de comida no recipiente.

Referências

APPINVENTOR. **Site oficial do App Inventor**. Disponível em: <http://appinventor.mit.edu/ > Acesso em 20.Out. 2013

ARDUINOECIA. **Arduino Uno e Ethernet Shield**. Disponível em: <http://www.arduinoecia.com.br/2013/06/ethernet-shield-wiznet-w5100-parte-1.html> Acesso em 10 Ago. 2013

ARDUINOTWEET. **Post messages from Arduino and Ethernet Shield**. Disponível em:
<http://arduino-tweet.appspot.com/> Acesso em: 20.Out.2013

ARDUINO-1. **Arduino Uno**. Disponível em:<http://arduino.cc/en/Main/ArduinoBoardUno> Acesso em 20 Out. 2013

ARDUINO-2. **Arduino Ethernet Shiled**. Disponível em: <http://arduino.cc/en/Main/ArduinoEthernetShield> Acesso em 20 Out. 2013

BOEIRA, Marcelo. 2013. **O que é Arduino?** Disponível em:
< http://blog.marceloboeira.com/arduino/o-que-e/> Acesso em: 22 Out .2013

BORTOLUZZI, Matias. Blog SRA Engenharia, **Histórico da automação residencial**,2013 Disponível em:<http://sraengenharia.blogspot.com.br/2013/01/historico-da-automacao-residencial_10.html>. Acesso em: 10 Set. 2013

BRITES, F.G & SANTOS, V.P.A,. **Motores de Passo.** Universidade Federal Fluminense.Curso de engenharia de Telecomunicações. Programa de Educação Tutorial.Niterói,RJ,2008.

CANYOUSEEME. **Open Port Check Too**l. Disponível em: <http://www.canyouseeme.org/> Acesso em 20.Out.2013

CHECKIP. **Indentify IP Adress**. Disponível em: <http://checkip.dyndns.com/>. Acesso em: 20.Out. 2013.

CRUZ, Renan Pontes. **Desenvolvimento de um Sistema de Supervisão Via Web Aplicado à Automação Residencial.** Universidade Federal do Rio Grande do Norte. Trabalho de Conclusão de Curso.Natal, RN, 2009.

DAILYMAIL, 2012. **Meet the dog fed his 'Tweet Treats' by contraption controlled remotely by Twitter.** Disponível em: <http://www.dailymail.co.uk/sciencetech/article-2110427/IT-geek-invents-feeds-dog-Twitter.html#ixzz2jKP3OGDR> Acesso em: 03.Ago.2013

DATASHEET W5100. Disponível em: (<https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf>). Acesso em 20.Out.2013

DIPOL. **Funcionamento do DDNS**. Disponível em: <http://www.dipol.pt/o_que_e_ddns_dynamic_domain_name_system_e_como_usa-lo__bib93.htm> Acesso em 15. Jun. 2013

EUZÉBIO, M. V.M. & MELLO, E. R. **Droidlar- Automação Residencial através do celular Android.** Sistemas de Telecomunicações, Instituto Federal de Santa Catarina.São José, SC, 2011.

INFOWESTER,2013. **O que é DNS?.** Disponível em: < http://www.infowester.com/dns.php>. Acesso em: 15 Out. 2013

JONES, Douglas W. **Control of Stepping Motors – A Tutorial**.The University Of Iowa .Department Of Computer Science, 1998. Disponível em :<,http://homepage.cs.uiowa.edu/~jones/step/types.html>Acesso em 05.Out.2013

JOSEMATM. **Actualiza tu DDNS desde Arduino**. Disponível em: <http://www.josematm.com/actualiza-tu-ddns-desde-arduino-dyndns/> Acesso em: 09 Set.2013

KEMPER. **Entendendo motores de robôs**. Disponível em: <http://www.kemper.com.br/wordpress/2011/06/30/red-%E2%80%93-entendo-os-motores-do-robo-parte-3/> Acesso em: 12.Out.2013

LADYADA. **Sensor Pir**. Disponível em:<http://www.ladyada.net/learn/sensors/pir.html> Acesso em 20. Out. 2013

MCROBERTS, Michael. **Arduino Básico.** Edição original em Inglês publicada pela Apress Inc., Copyright © 2010 pela Apress, Inc.. Edição em Português para o Brasil copyright © 2011 pela Novatec Editora.

NO-IP. **Alocação de servidores DDNS**. Disponível em: <www.noip.com> Acesso em 13 Abr. 2013

REGISTRO. **Hierarquia de Domínios**. Disponível em: <www.registrodedominios.net.br> Acesso em 07 Set. 2013

SILVA, B.C.R & CÂNDIDO, L.A.A. **Sistema de Controle Residencial baseado na plataforma Arduino**. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Instituto Unificado de Ensino Superior Objetivo, Goiânia, 2011.Disponível em: <http://pt.scribd.com/doc/80802432/Sistema-de-Controle-Residencial-Baseado-Na-Plataforma-Arduino> Acesso em: 15 Out. 2013

STARDOT. **Câmera IP.** Disponível em: <http://www.stardot.com.au/accessories/webcams/ipcams.html> Acesso em: 13.Ago.2013

TECHMUNDO, 2013. **O que é TCP/IP?**. Disponível em:<http://www.tecmundo.com.br/o-que-e/780-o-que-e-tcp-ip-.html> Acesso em: 13.Ago. 2013

WOLBER et al. **App Inventor- Create Your Own Android App**. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. First Edition, 2011.

Apêndice A - Código Arduino

```
#include <SPI.h>

#include <Ethernet.h>

#include <Twitter.h>

#include <EEPROM.h>


byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress ip(192,168,0,199); //ip que o Arduino terá na Intranet

IPAddress gateway(192,168,0,1); //ip do roteador



EthernetServer server(8888); //porta de acesso ao servidor

EthernetClient client;


// Token para Tweetar ( http://arduino-tweet.appspot.com/)

Twitter twitter("1673941976-xQI9XicACAYPxMMX3g9jJdV8X5dWQSzxmfXVT");



char msg[] = "@LucasBeghini o alarme foi disparado"; // mensagem quando se dispara o alarme

char msg2[]="@LucasBeghini, atualize o ip no aplicativo!";

char servidorIP[] = "checkip.dyndns.org"; // site de deteccão do ip publico


IPAddress ipAtual;

IPAddress ipUltimo;

IPAddress ipnulo (0,0,0,0);


int buffalarme=0;   // controla se alarme está ligado ou desligado

int buffdog=0;  //controla se o compartimento está aberto ou fechado

int i=0,j=0,passos=10;

long int cont=1000000; // variavel contadora para requisição de ip publico

int motor[]= {3,7,8,9}; // vetor que controla saídas para o motor de passo


union IPAddressConverter { // conversor para uint8 do ip

  uint32_t ipInteger;

  uint8_t ipArray[4];

};


void setup()

{

  pinMode(2,INPUT); //sensor alarme

  pinMode(4, OUTPUT); // saída lâmpada

  pinMode(5, OUTPUT);// alimentação alarme

  pinMode(6, OUTPUT);//sirene alarme


  pinMode(3, OUTPUT);// saídas motor de passo

  pinMode(7, OUTPUT);

  pinMode(8, OUTPUT);

  pinMode(9, OUTPUT);


  digitalWrite(2, LOW);//Funciona como resistor de pull-down para o PIR


  Serial.begin(9600);

  Ethernet.begin(mac, ip, gateway);

  server.begin();


  delay(1000);


  for (byte n = 0; n <= 3; n++) ipUltimo[n] = EEPROM.read(n); // leitura da EEPROM

  Serial.print("O IP armazenado na EEPROM eh: ");

  Serial.println(ipUltimo);


}


void loop()
```

```
{


  int sensorpir = digitalRead(2);// leitura do sensor de movimento


  if (sensorpir==HIGH){

    Serial.println("alarme disparado");

    Serial.println(buffalarme);

    if(buffalarme==0){


   digitalWrite(6, HIGH); // dispara sirene


   if (twitter.post(msg)) { // posta notificação no Twitter de disparado

    int status = twitter.wait();

    if (status == 200) {

      Serial.println("OK.");

    } else {

      Serial.print("failed : code ");

      Serial.println(status);

    }

  } else {

    Serial.println("connection failed.");

  }

  buffalarme=1;

}


   delay(2000);

    }


    if(cont==0) {

      cont=1000000;

ipAtual = descobreIP();  // descobre o ip publico da rede onde esta o Arduino


if (ipAtual!=ipnulo){ // se o IP não for nulo

if (ipAtual != ipUltimo) { // se o ip mudar


      for (byte n = 0; n <= 3; n++) {

      EEPROM.write(n, ipAtual[n]);}


      ipUltimo = ipAtual;


     IPAddressConverter ipAddress;  // transforma uint32 para uint8 para enviar por Twitter

      ipAddress.ipInteger = ipAtual;

      char buf[16];

      sprintf(buf, "%d.%d.%d.%d", ipAddress.ipArray[0], ipAddress.ipArray[1], ipAddress.ipArray[2], ipAddress.ipArray[3]);// imprime o IP em uma string


    if (twitter.post(msg2)) { // posta no Twitter a msg2

    int status = twitter.wait();

    if (status == 200) {

      Serial.println("OK.");

    } else {

      Serial.print("failed : code ");

      Serial.println(status);

    }

  } else {

    Serial.println("connection failed.");

  }

  delay(10000);

    if (twitter.post(buf)) { // posta no Twitter o novo Ip

    int status = twitter.wait();

    if (status == 200) {

      Serial.println("OK.");

    } else {

      Serial.print("failed : code ");

      Serial.println(status);

    }

  } else {

    Serial.println("connection failed.");

  }


     Serial.print(ipAtual);


      }


  else {

    Serial.println("O IP não mudou");
```

```
    Serial.println(ipUltimo);
    Serial.println(ipAtual);
  }
}
    }
  Principal();
 }
void Principal()
{
  cont--;
  EthernetClient client =
server.available();
  if (client) {
    boolean newLine = true;
    String line = "";
while (client.connected() &&
client.available()) {
       char c = client.read();
      if (c == '\n' && newLine) {
         client.println("HTTP/1.1
200 OK");
         client.println("Content-
Type: text/html");
         client.println();

client.println(digitalRead(4));  // lê
as portas do Arduino ou
variaveis para feedback do
aplicativo

client.println(digitalRead(5));

client.println(digitalRead(6));
         client.println(buffdog)
         }
         if (c == '\n') {
           newLine = true;
           evaluateLine(line);
           line = "";
         }
         else if (c != '\r') {
           newLine = false;
           line += c;
         }
    }
    evaluateLine(line);

    delay(1);
    client.stop();
  }
}

void evaluateLine(String line)
{
  if (line.startsWith("tag", 0)) {
    String instrucao =
line.substring(4, 11); // Pega as 7
letras da instrução vinda do App
    Serial.print (line);
   if (instrucao == "liglamp") {
//liga lampada
       digitalWrite(4, HIGH);
    }
    if (instrucao == "deslamp") {
//desliga lampada
      digitalWrite(4, LOW);
    }
    if (instrucao == "ativala") {
//ativa alarme
      digitalWrite(5, HIGH);
      delay(3000);
      digitalWrite(6, LOW);
    }
     if (instrucao == "desaala") {
//desativa alarme
      digitalWrite(5, LOW);
      delay(3000);
    }
     if (instrucao == "deslala") {
//desliga alarme
      digitalWrite(6, LOW);
      buffalarme=0;
    }

    if (instrucao == "aliment") {  //
abre/fecha compartimento de
ração

      if (buffdog==0){
       abrir();
       buffdog=1;
      }
       else {
       fechar();
       buffdog=0;
```

```
      }


    j=0;
   }
    }
}
 void abrir() // abre
compartimento
 {
  while (j<passos) {
  for(i=0;i<4;i++){

    digitalWrite(motor[i], HIGH);
    delay(500);
    digitalWrite(motor[i], LOW);
  }
  j++;
  }
}
  void fechar() //fecha
compartimento
 {
  while (j<passos) {
  for(i=3;i>=0;i--){

    digitalWrite(motor[i], HIGH);
    delay(500);
    digitalWrite(motor[i], LOW);
  }
```

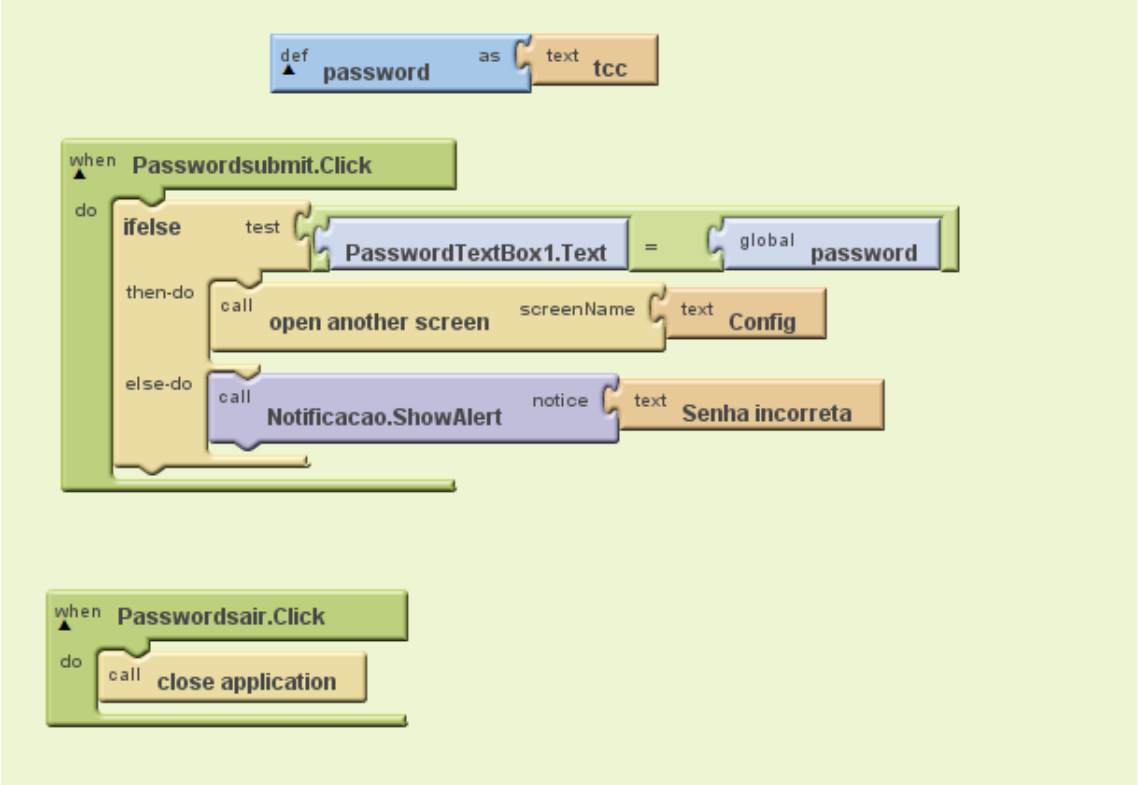
```
  j++;
 }
 }

 IPAddress descobreIP() { //
descobre o ip da rede do
Arduino

 EthernetClient client;
 String webIP;
 int desde, ateh;

 if (client.connect(servidorIP,
80)) {

  client.println("GET /
HTTP/1.0");
  client.println();

  webIP = "";
 } else {
  Serial.println("Falha");
 }
 while (client.connected()) {
  while (client.available()) {

webIP.concat((char)client.read())
;
  }
 }

 client.stop();
```

```
 desde =
webIP.indexOf("Address: ") + 9;
 ateh =
webIP.indexOf("</body>");

 return
ipAIPAddress(webIP.substring(d
esde, ateh));

}

IPAddress ipAIPAddress(String
ipEnCadena){   // Interpreta o
HTML recebido e devolve
somente o numero do IP

 IPAddress ipBytes;
 char digitoIP[4];
 byte cursorDigito = 0;
 byte cursorIP = 0;

 for (byte n = 0; n <
ipEnCadena.length(); n++){
  if (ipEnCadena.charAt(n) != '.')
{
    digitoIP[cursorDigito] =
ipEnCadena.charAt(n);
    cursorDigito++;
  } else {
    digitoIP[cursorDigito +1] =
'\n';
    ipBytes[cursorIP] =
atoi(digitoIP); // converte string
em numero inteiros
```

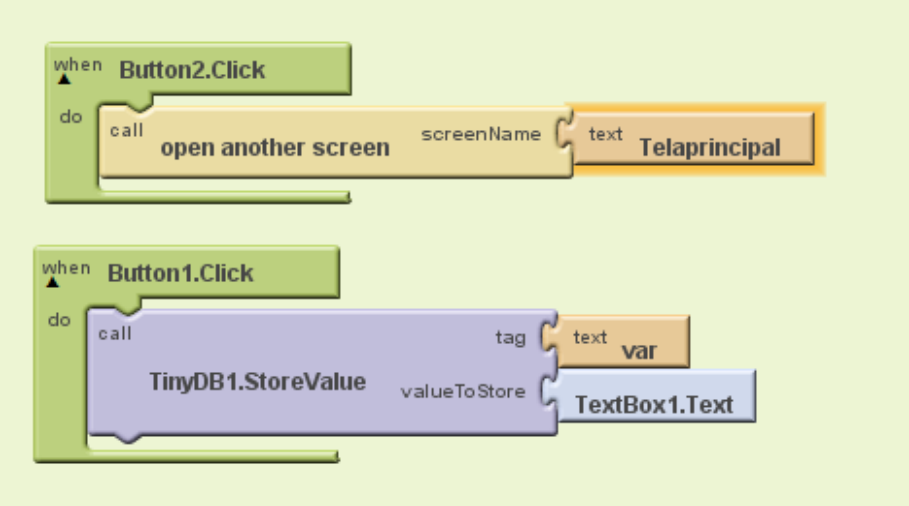
```
      cursorDigito = 0;

      memset(digitoIP, 0,
sizeof(digitoIP));

      cursorIP++;

    }

  }

  digitoIP[cursorDigito +1] = '\n';

  ipBytes[cursorIP] =
atoi(digitoIP);



  return ipBytes; //devolve IP em
Bytes

}
```

Apêndice B - Projeto no *App Inventor*

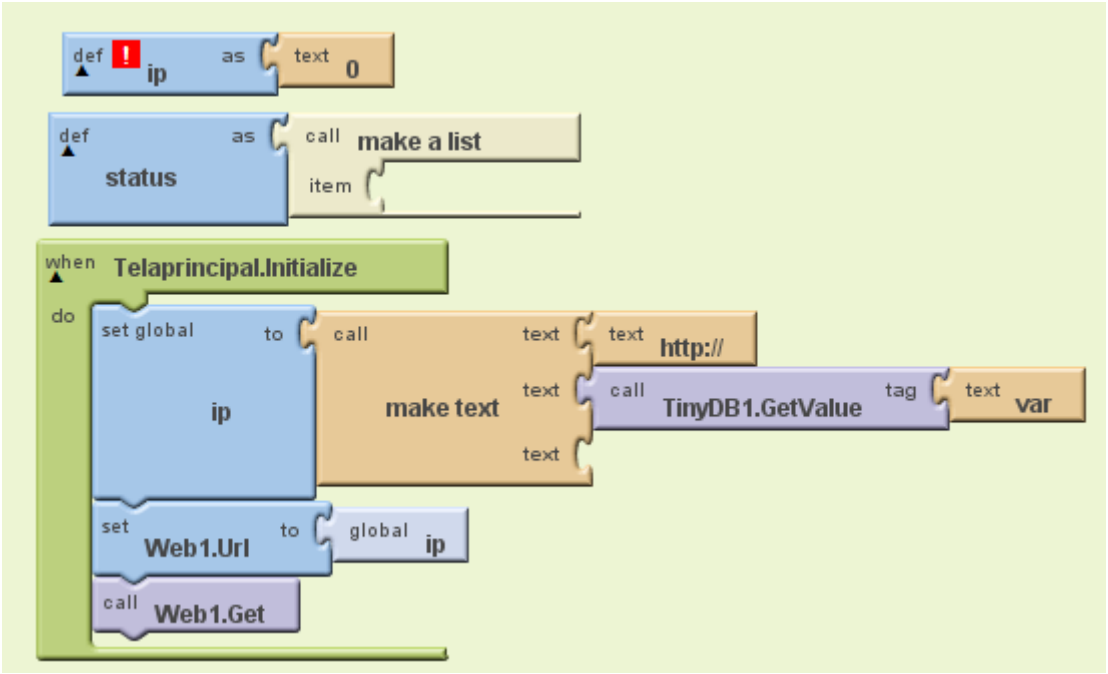
Tela inicial:

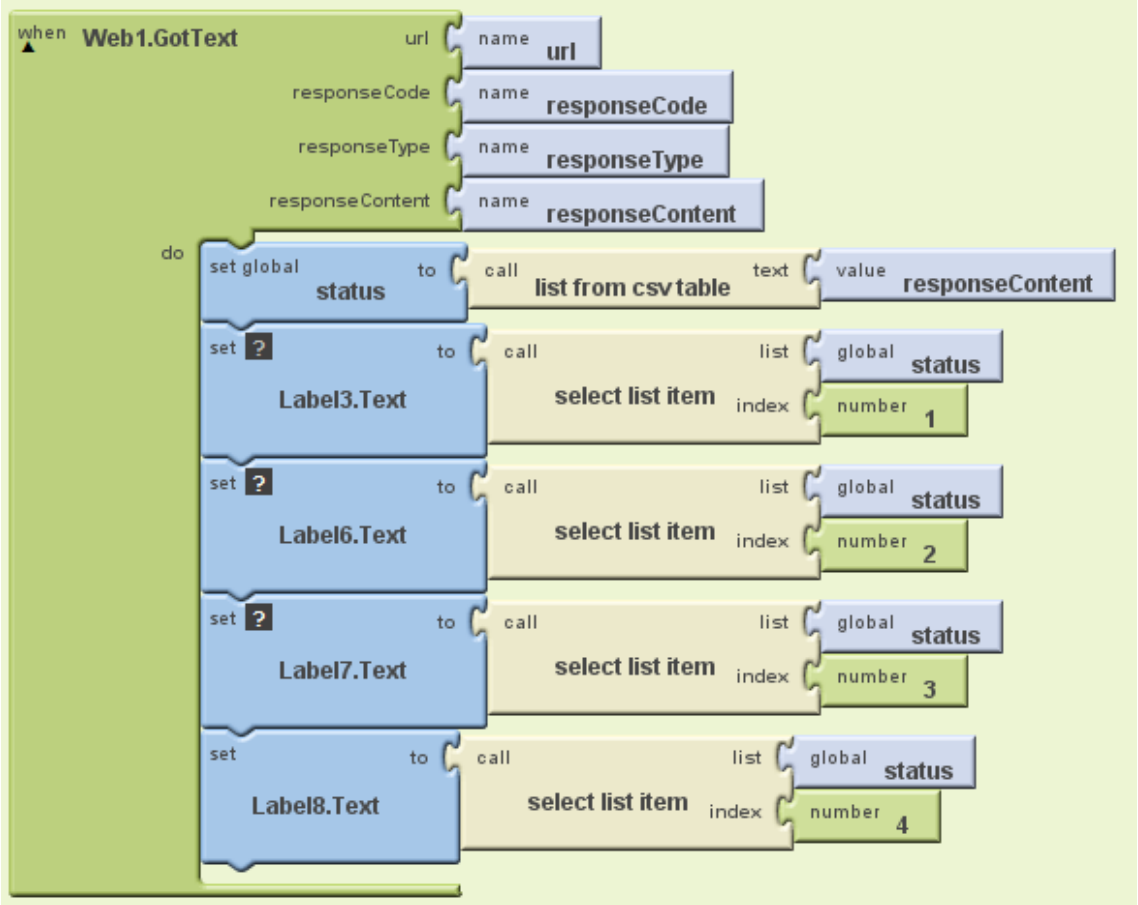


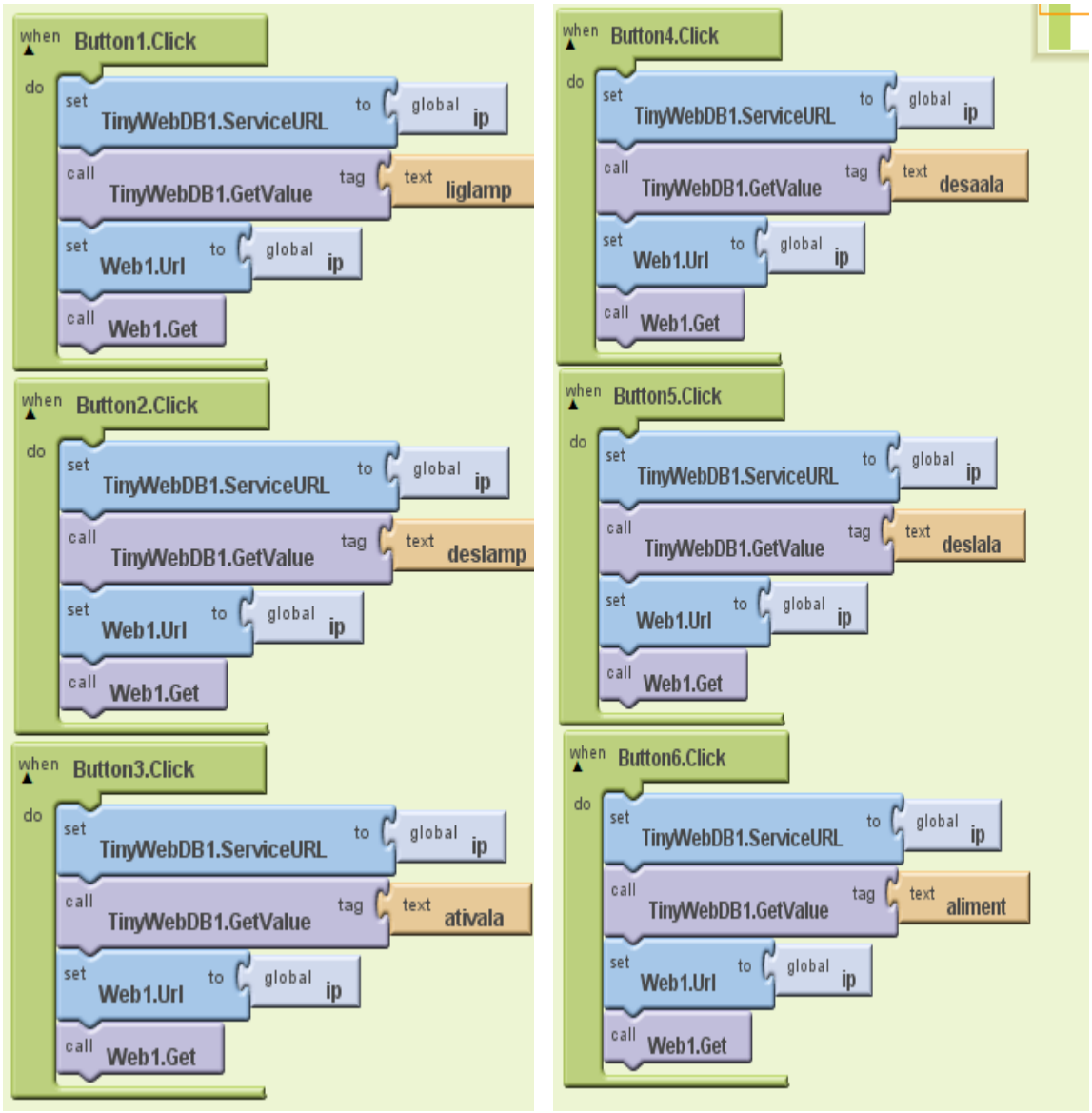
Tela Config:

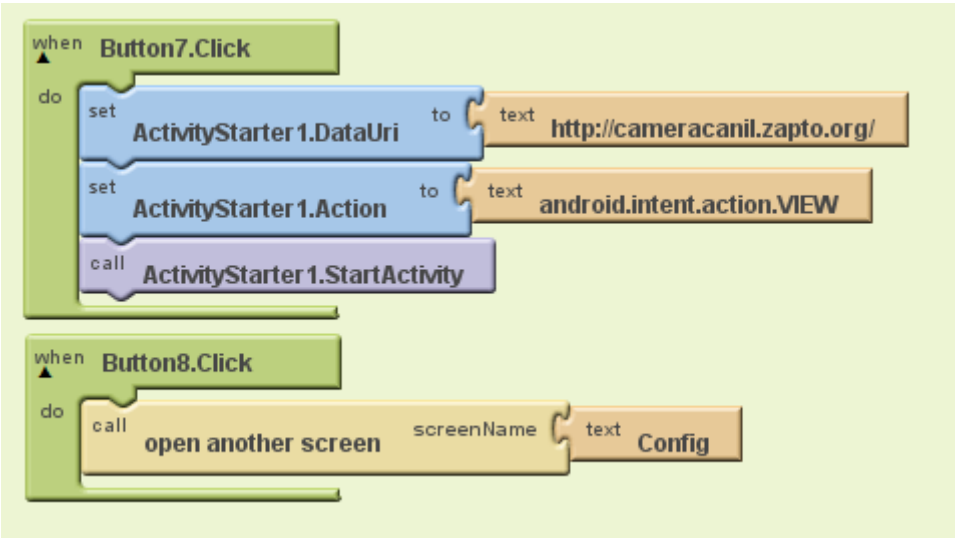


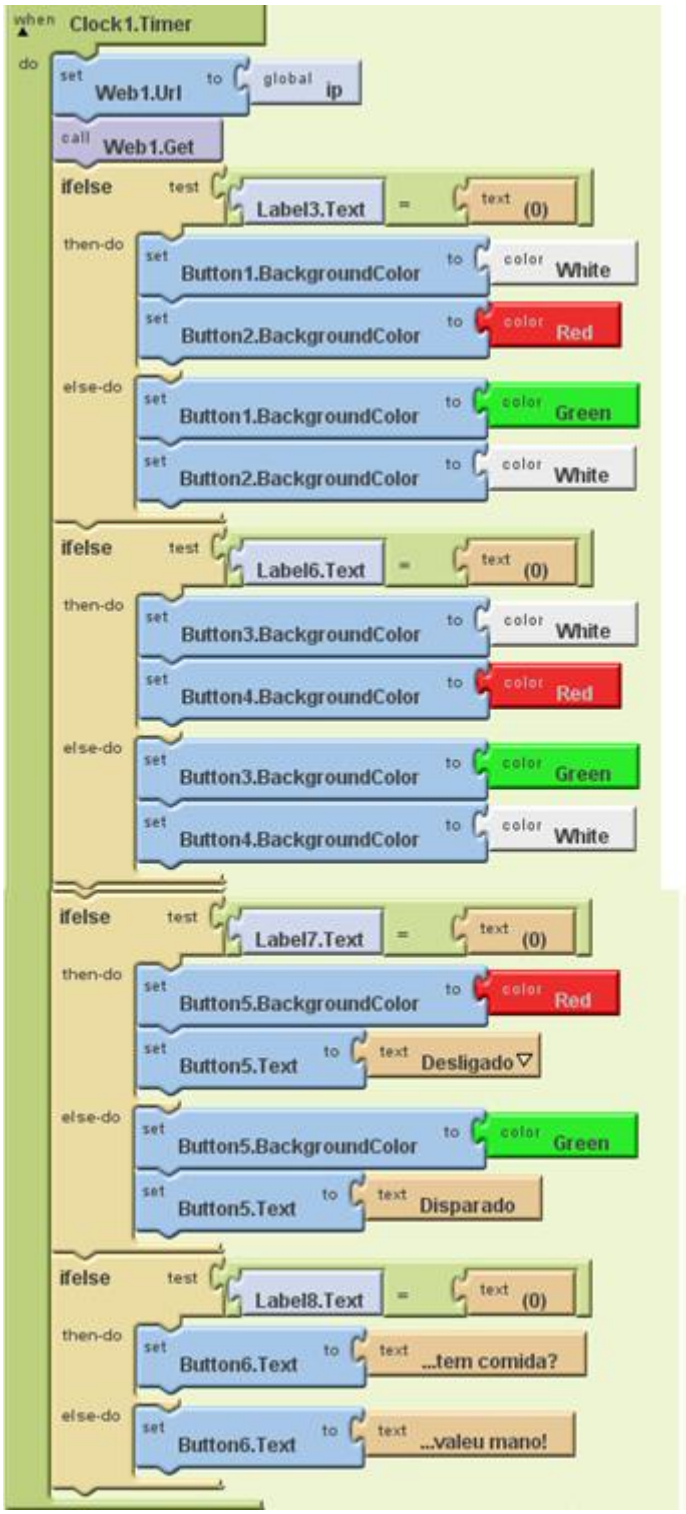
Tela Principal:






when Button1.Click
do set TinyWebDB1.ServiceURL to global ip
call TinyWebDB1.GetValue tag text liglamp
set Web1.Url to global ip
call Web1.Get
when Button2.Click
do set TinyWebDB1.ServiceURL to global ip
call TinyWebDB1.GetValue tag text deslamp
set Web1.Url to global ip
call Web1.Get
when Button3.Click
do set TinyWebDB1.ServiceURL to global ip
call TinyWebDB1.GetValue tag text ativala
set Web1.Url to global ip
call Web1.Get
when Button4.Click
do set TinyWebDB1.ServiceURL to global ip
call TinyWebDB1.GetValue tag text desaala
set Web1.Url to global ip
call Web1.Get
when Button5.Click
do set TinyWebDB1.ServiceURL to global ip
call TinyWebDB1.GetValue tag text deslala
set Web1.Url to global ip
call Web1.Get
when Button6.Click
do set TinyWebDB1.ServiceURL to global ip
call TinyWebDB1.GetValue tag text aliment
set Web1.Url to global ip
call Web1.Get


when Button7.Click
do set ActivityStarter1.DataUri to text http://cameracanil.zapto.org/
set ActivityStarter1.Action to text android.intent.action.VIEW
call ActivityStarter1.StartActivity
when Button8.Click
do call open another screen screenName text Config


when Clock1.Timer
do
set Web1.Url to global ip
call Web1.Get
ifelse test Label3.Text = text (0)
then-do
set Button1.BackgroundColor to color White
set Button2.BackgroundColor to color Red
else-do
set Button1.BackgroundColor to color Green
set Button2.BackgroundColor to color White
ifelse test Label6.Text = text (0)
then-do
set Button3.BackgroundColor to color White
set Button4.BackgroundColor to color Red
else-do
set Button3.BackgroundColor to color Green
set Button4.BackgroundColor to color White
ifelse test Label7.Text = text (0)
then-do
set Button5.BackgroundColor to color Red
set Button5.Text to text Desligado
else-do
set Button5.BackgroundColor to color Green
set Button5.Text to text Disparado
ifelse test Label8.Text = text (0)
then-do
set Button6.Text to text ...tem comida?
else-do
set Button6.Text to text ...valeu mano!

Anexo A - Datasheet W5100

WIZnet

Features

- Support Hardwired TCP/IP Protocols : TCP, UDP, ICMP, IPv4 ARP, IGMP, PPPoE, Ethernet
- 10BaseT/100BaseTX Ethernet PHY embedded
- Support Auto Negotiation (Full-duplex and half duplex)
- Support Auto MDI/MDIX
- Support ADSL connection (with support PPPoE Protocol with PAP/CHAP Authentication mode)
- Supports 4 independent sockets simultaneously
- Not support IP Fragmentation
- Internal 16Kbytes Memory for Tx/Rx Buffers
- 0.18 μm CMOS technology
- 3.3V operation with 5V I/O signal tolerance
- Small 80 Pin LQFP Package
- Lead-Free Package
- Support Serial Peripheral Interface(SPI MODE 0, 3)
- Multi-function LED outputs (TX, RX, Full/Half duplex, Collision, Link, Speed)

Block Diagram