Internet das Coisas

Cloadoaldo Basaglia da Fonseca
Douglas Lohmann
Marco Aurélio Graciotto Silva
Paulo Cesar Gonçalves

Este trabalho foi financiado pela Fundação Araucária - Apoio ao Desenvolvimento Científico e Tecnológico do Paraná, por meio do Edital Redes Digitais de Cidadania do Estado do Paraná (Ministério das Comunicações), aprovado em 2013. Foi desenvolvido por alunos e professores da Universidade Tecnológica Federal do Paraná (UTFPR), campus Campo Mourão

Uma versão online desse material está disponível em https://github.com/lohmanndouglas/Iot-Compute-Voce-Mesmo.git

Sumário

Capítulo 1

Uma Visão Geral

Computadores pessoais e smartphones formam uma rede de dispositivos conectados a Internet. A questão agora é permitir que outros dispositivos tais como relógios, máquinas de lavar, geladeiras e demais objetos do nosso cotidiano possam conectar-se a rede e trocar informações. Esta fase em formação está introduzindo um novo paradigma chamado de Internet das Coisas (do inglês, Internet of Things - IoT), no qual pessoas, animais e coisas do nosso cotidiano estão conectados à rede e interagirem entre si.

A Internet das Coisas mudará tudo, inclusive nós mesmos. Considerando o impacto que Internet já causou na comunicação, nos negócios, na ciência, no governo e na educação, percebemos claramente que a Internet é uma das mais importantes e poderosas invenções de toda a história humana [**?**]. Devido ao desenvolvimento das tecnologias de informação, principalmente da Internet, podemos nos comunicar tranquilamente com qualquer parte do mundo, Desta forma, possuímos a oportunidade de conhecer muitas coisas - novas pessoas, culturas, sistemas políticos, desenvolvimento de cada país e muitas coisas mais - por meio de alguns cliques.

Podemos dividir a Internet em três fases. A primeira fase é

a Internet como uma rede de computadores. Na segunda fase, a Internet pode ser considerada uma rede de pessoas e comunidades e atualmente estamos vivendo a evolução para terceira fase, a Internet das Coisas (IoT). Nesta fase a rede passa a interligar vários tipos de objetos e dispositivos inteligentes do nosso cotidiano que vão interagir entre si e conosco [**?**]. Segundo [**?**], a ideia básica de IoT consiste na presença de uma diversidade de objetos que interagem e cooperam entre si a fim de atingir um objetivo comum. Para tal compartilham informações utilizando métodos de endereçamento único e protocolos de comunicação padronizados.

Este material apresenta os principais conceitos relacionados a Internet das Coisas e também apresenta uma atividade prática para implementação de IoT. Os próximos tópicos são referencias básicas para a construção da rede de sensores e a comunicação dos sensores com a Internet.

Capítulo 2

Materiais e Softwares Utilizados

Este capítulo apresenta uma breve descrição dos principais materiais e softwares necessários para implementação dos projetos propostos.

2.1 Arduino

Arduino [1] é uma plataforma de prototipagem de código aberto baseada na fácil utilização do software e hardware. As placas Arduino são capazes de efetuarem leitura de uma entrada (sensores) e transformar em uma de saída (Atuadores). O projeto Arduino nasceu no Ivrea Interaction Design Institute como uma ferramenta fácil para prototipagem rápida, destinado a estudantes sem conhecimento aprofundado em eletrônica e programação.

A plataforma Arduino possui uma IDE para a programação e para gravar códigos na placa, a IDE está possui suporte a

[1] https://www.arduino.cc/en/Guide/Introduction

Linux, Mac e Windows.

Existem diversas placas de hardware Arduino, sendo a mais comum a Arduino Uno. As placas diferem basicamente no microcontrolador embutido, no número de entradas/saídas, na frequência de processamento e entre outras configurações. Neste Material vamos utilizar o Arduino Uno por ser facilmente encontrado no mercado e apresentar baixo custo de aquisição se comparado com outras plataformas de hardware.

A conexão de novos componentes no Arduino é possível por meio de placas de expansão (*shields*), essa placas aumentam as funcionalidades do Arduino. Os *shields* mais conhecidos são o *shields* para controle de motores [2] e o Ethernet para comunicação do Arduino com a Internte como o da Figura 2.1.

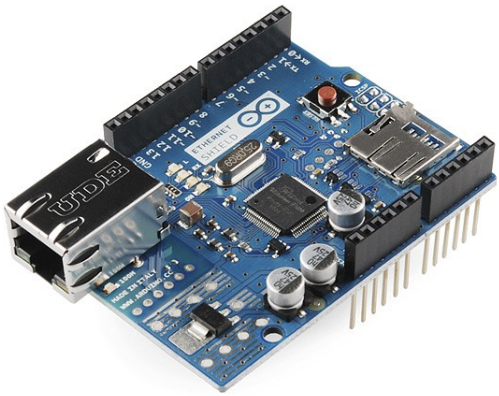

Figura 2.1: Shield para controle de motores

A comunicação do Arduino com *shields* é realizada pelo

[2]https://www.arduino.cc/en/Main/ArduinoMotorShieldR3

protocolo Serial Peripheral Interface (SPI) [3] é um protocolo de dados seriais síncronos utilizado em microcontroladores para comunicação entre o microcontrolador e um ou mais periféricos. Também pode ser utilizado entre dois microcontroladores. A comunicação SPI sempre tem um master. Isto é, sempre um será o master e o restante será slave. Por exemplo, o Arduino é o master e os outros periféricos são slaves. Esta comunicação contém 4 conexões:

- MISO (Master IN Slave OUT) - Dados do Slave para Master;
- MOSI (Master OUT Slave IN) - Dados do Master para Slave;
- SCK (Serial Clock) - Clock de sincronização para transmissão de dados entre o Master e Slave;
- SS (Slave Select) - Seleciona qual Slave receberá os dados.

2.1.1 Arduino IDE

Para programação no arduino, utiliza-se a linguagem Wiring. O ambiente de programação oferece recursos que facilitam a criação de aplicações e sua gravação no dispositivo.

Os projetos (Sketches) são escritos na linguagem Wiring e salvos com a extensão .ino, possuem a estrutura:

Listing 2.1: Estrutura código Arduino

```
1 // Inclusao de bibliotecas
2 #include ...
3 //Declaracao de variaveis
4 void setup() {...}
5 void loop () {...}
6 //Funcoes auxiliares
```

[3] https://www.arduino.cc/en/Reference/SPI

A função setup() é o código de inicialização dos componentes, enquanto a função loop é o laço principal do programa, que contém o código que será executado repetidamente.

2.1.2 Arduino Uno

O Arduino Uno [4] opera com uma velocidade de clock de 16 MHz, possui 14 pinos de entrada e saída digitais e 6 pinos de entrada e saída analógica, memória flash 32 KB (0.5 KB usados pelo Bootloader), memória SRAM de 2 KB e 1 KB de memória EEPROM.

A placa pode ser alimentada pela conexão USB ou por uma fonte de alimentação externa. Para alimentação externa é utilizado um conector Jack com positivo no centro, sendo que a placa suporta alimentação de 6 à 20 volts. Porém, é recomendado que a fonte de alimentação externa possua tensão entre 7 e 12 volts.

A Figura 2.2 ilustra o Arduino Uno utilizado neste projeto, ele tem 14 pinos de entradas/saídas digitais. Alguns desses pinos possuem funções especificas como PWM (pinos 3, 5, 6, 9, 10 e 11), comunicação serial (pinos 0 e 1) e interrupção externa (pinos 2 e 3). Para interface com o mundo analógico, a placa Arduino UNO possui 6 entradas, onde cada uma tem a resolução de 10 bits.

Figura 2.2: Arduino Uno

[4]https://www.arduino.cc/en/Main/ArduinoBoardUno

2.2 Radio RF - nRF24l01

Nesta apostila vamos utilizar o módulo de rádio frequência nRF24l01 [5] fabricado pela Nordic, este módulo trabalha na frequência de 2.4 GHz.

A conexão é realizada por um conector de 8 pinos muito próximos uns dos outros, o que impossibilita a conexão direta do módulo com a protoboard, sendo assim a conexão com o módulo pode ser feita com jumpers macho-femea, outra possibilidade é construir um shield para adaptação, para que o nRF24l01 encaixe na protoboard.

O alcance do módulo varia de 10 metros em ambiente fechado à 50 metros em ambiente aberto. Uma outra vantagem é que um mesmo módulo pode atuar como emissor ou receptor, apenas realizando-se uma configuração por software. Sua tensão de alimentação é de 1,9 à 3.6V, e os pinos de sinal podem trabalhar normalmente com nível de sinal de 5V.

Existe uma versão do módulo com antena externa, essa versão possibilita distancia maior de comunicação entre os módulos, porém tem preço mais maior e não é tão compacto quanto o módulo com antena embutida.

Figura 2.3: Rádio nRF24L01

[5] http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01

2.3 Sensores e Atuadores

Esta seção é uma breve descrição dos principais sensores e atuadores utilizados para realizar os projetos propostos nesse material.

2.3.1 DTH11

O DTH11 [6] é um sensor de baixo custo para a medição de temperatura e umidade do ambiente. Sua faixa de medição de temperatura vai de 0° a 50° Celsius, com 2% de margem de erro. Já a medição de umidade pode variar de 20% até 90% com precisão de 5%.

O sensor possui 4 pinos: um pino para o GND(ground), um para a alimentação(5V), um pino para envio dos dados que é conectado a uma entrada digital do Arduino e um pino que não é utilizado.

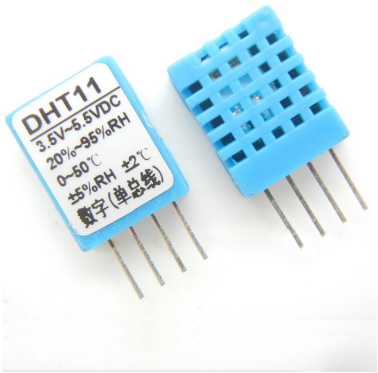

Figura 2.4: Sensor DTH11

[6] https://learn.adafruit.com/dht

2.3.2 LDR

O LDR (do inglês, Light Dependent Resistor), ou Resistor dependente de Luz, é uma fotorresistência, ou seja, um resistor cuja sua resistência varia de acordo com a intensidade da luz que incidir sobre ele.

Utilizando um multímetro pode-se medir a resistência de um LDR quando exposto a uma determinada intensidade de luz. Basicamente um LDR vai ter sua resistência máxima quando estiver em completa escuridão e mínima quando uma luz muito brilhante estiver incindindo sobre ele.

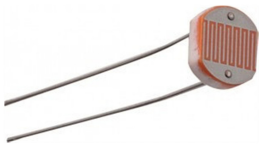

Figura 2.5: Sensor LDR

2.3.3 Relé

O relé é um dispositivo eletromecânico capaz de desligar ou ligar outros dispositivos. Basicamente, o relé é acionado quando uma corrente elétrica passa a percorrer as espiras da bobina do mesmo, criando assim a um campo magnético que atrai ou repele uma alavanca responsável por ativar ou desligar o outro componente ligado ao relé.

Figura 2.6: Atuador relé

2.3.4 Sensor de Movimento (PIR)

O sensor de movimento (PIR) [7]é basicamente feito de uma material piroelétrico, ou seja, seu potencial elétrico varia de acordo com a temperatura, tornando-o capaz de detectar alguns níveis de radiação infravermelha. Ele é construído em duas metades e as duas são conectadas de forma que a diferença de potencial entre elas seja interpretada como um nível alto ou baixo, fazendo assim a detecção de movimento.

Figura 2.7: Sensor de Movimento PIR

[7]https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/

2.3.5 LED

Light Emitting Diode(LED), ou diodo emissor de luz, é utilizado para emissão de luz em locais onde lampadas não são viáveis, por exemplo em produtos da microeletrônica como sinais de avisos. Existem também LEDs de tamanhos maiores, como os utilizados em sinais de transito. Uma característica do LED é seu baixo consumo de energia, sendo uma alternativa viável para iluminação de ambientes. Outra vantagem é que sua durabilidade é maior que as outras formas de emissão de luz presentes no mercado. Na maioria dos projetos vamos utilizar o LED como atuador para emitir sinais de aviso. Mais detalhes sobre o funcionamento do LED estão disponíveis em http:electronics.howstuffworks.comled.htm

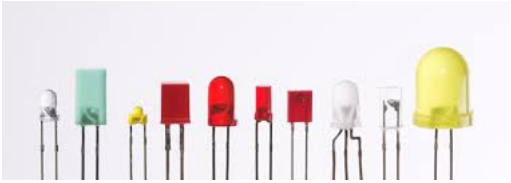

Figura 2.8: LEDs

2.4 Protoboard

É uma placa com uma matriz de furos e conexões condutoras utilizadas para a prototipação de circuitos eletrônicos.

A Figura 2.9 ilustra uma protoboard. Essa placa tipicamente possui trilhas conectas na vertical que possibilitam a ligação entre componentes e trilhas isoladas na horizontal.

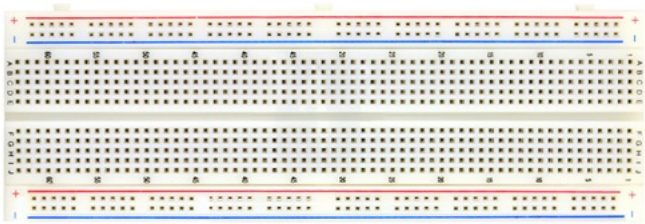

Figura 2.9: Protoboard

2.5 Conectar todas as partes

Para conectar os sensores e atuadores a placa do Arduino, é utilizado cabo jumper. Podem ser encontrado em 3 combinações: macho-macho, macho-femea e femea-femea.

A Figura 2.10 representa jumpers macho.

Figura 2.10: Jumper Macho

A Figura 2.11 representa jumpers fêmea.

Figura 2.11: Jumper Fêmea

2.6 Conectando o Arduino ao Rádio

O rádio nRF24l01+ se comunica com Arduino via interface SPI. O rádio deve ser alimentado com uma tensão de 3.3 volts.

Como o radio possui conector de 8 pinos não é possível conecta-lo a protoboard. Então devemos utilizar conectores macho-fêmea, como ilustra a Figura 2.12, para fazer ligação ou construir um shield para adaptação do modulo.

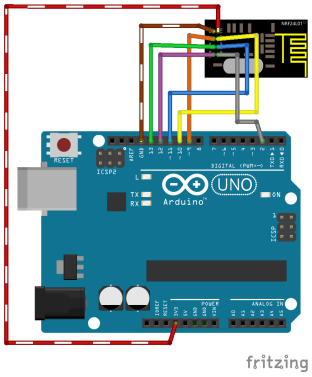

Figura 2.12: Conexão rádio e Arduino Uno

2.7 Biblioteca Mysensors

Mysensors [8] é uma API que fornece um conjunto de protocolos e rotinas para comunicação entre o Arduino, o Radio transmissor e sensores. Com esta biblioteca é possível criar uma abstração de algumas camadas de hardware e software evitando que o usuário tenha que implementar rotinas de baixo nível e protocolos para realizar a comunicação entre o Arduino e Modulo de radio frequência. A biblioteca fornece códigos de exemplos e uma estrutura de rede já implementada para Internet das Coisas.

2.7.1 Aplicações da biblioteca MySensors

A biblioteca MySensors fornece uma infinidade de projetos com exemplos de sensores e atuadores já implementados. Algumas possibilidades de aplicação da biblioteca são:

- Pequenas automações residenciais tais como um portão de garagem automático que abre quando carro se aproxima da entrada ou quando ativado pelo seu *smartphone*;
- Coletar a umidade do ambiente dentro da casa para controlar a ventilação;
- Criar uma fechadura inteligente para um determinado comodo ou armário;
- Criar um dispositivo que avisa através de uma mensagem em seu *smartphone* quando a temperatura do frezzer subir caso alguém tenha deixado a porta aberta;
- Criar uma identificação única para seu cachorro para que somente ele possa entrar em sua casa.

[8] http://www.mysensors.org/

2.7.2 Instalar Mysensors

MySensors é uma biblioteca que funciona integrada com a API do Arduino, para instalação basta adiciona-lá a pasta *libraries* do Arduino IDE.

Capítulo 3

Infraestrutura da Internet das Coisas

3.1 Componentes envolvidos

3.1.1 O cérebro

O Arduino foi a escolha para esse projeto, pois tem consumo de energia baixo, é fácil de programar,muitas bibliotecas já estão implementadas e são livres para uso, além disso, é um hardware livre.

Apesar de não ter um poder computacional muito alto, é ideal para uso nesse projeto. Além disso, ele tem pinos (22 no total) que são ideais para conectar sensores e botões. Outra característica é que o Arduino não possui sistema operacional.

3.1.2 O rádio

Com a necessidade de comunicação e coleta de dados dos sensores e devido a possível distancia entre eles, é necessário uma conexão sem fio. Para tais fins, é utilizado um pequeno

rádio, nesse caso, o modelo nRF24l01, que tem baixo consumo de bateria e baixo preço.

3.1.3 Software

Além dos componentes de hardware vamos utilizar alguns softwares, na maior parte bibliotecas com alguns exemplos de sensores já implementados.

3.2 Construção da Rede de Sensores e Atuadores

Utilizando a biblioteca e as plataformas descritas no Capítulo 2, é possível configurar uma infraestrutura para a rede IoT, como ilustra a 3.1. A rede implementada possui basicamente três componentes: controlador, gateway e nós finais, que são nós sensores e atuadores. Os nós sensores e atuadores são responsáveis pela interação com o ambiente, seja pela coleta de informações por meio de sensores, emissão de sinais de alerta ou ativação de certos dispositivos por meio de atuadores, por exemplo o ar condicionado ou uma lâmpada.

Os principais componentes de uma rede de sensores são apresentados nas próximas subseções.

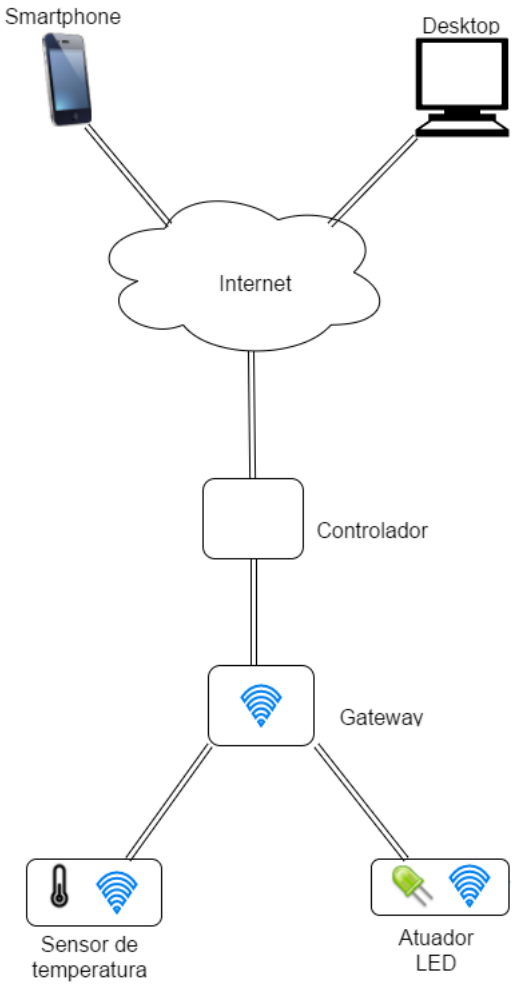

Figura 3.1: Infraestrutura iot

A Figura 3.1 ilustra a infraestrutura de uma rede de sensores e atuadores, pode-se perceber a presença do nó gateway conectando os nós sensores e o controlador. Embora na imagem a ligação entre os dispositivos é representada por linhas, na prática a ligação entre os nós na maioria das vezes não é feita por jumper e sim via rádio frequência.

3.2.1 Nó sensor ou atuador

Esse nó realiza a leitura de sensores e pode também funcionar como um nó atuador, enviando e recebendo dados do gateway. Esse nó pode funcionar em modo Sleep para economizar bateria.

3.2.2 Nó repetidor

Esse nó só é necessário quando os nós sensores e gateway não conseguem se comunicar, devido à distancia em que estão localizados, por esse motivo não está representado na Figura 3.1. Esse nó tem como função repetir as dados para outros nós a fim de aumentar a distancia de comunicação entre os nós. Em muitas aplicações esse nó não esta presente.

3.2.3 Nó gateway

O gateway atua na ligação entre o controlador e rede de rádios. Ele traduz as mensagens do rádio para um protocolo que pode ser entendido por um controlador.

Na biblioteca MySensors exite tres implementações de Gateway, o Ethernet Gateway, SerialGateway e MQTTGateway. Nesse material vamos utilizar o SerialGateway pela facilidade de implementação e também pela compatibilidade com o controlador escolhido.

3.2.4 Controlador

O controlador pode realizar as seguintes funções:

- Enviar parâmetros de configuração para os sensores na rede de rádio (tempo e identificadores de sensores únicos);

- Acompanhar os dados mais recentes enviados pelos sensores e atuadores;
- Fornecer informações de status de volta para sensores e atuadores; por exemplo o estado atual (on / off / loadLevel) para uma luz;
- Fornecer controles de interface do usuário para atuadores;
- Executa horários predefinidos ou cenas; por exemplo. ao pôr do sol acender as luzes do jardim.

Neste material vamos utilizar como controlador o Raspberry e o framework Pimatic.

3.3 Entendendo o protocolo serial MySensors versão 1.5

O protocolo utilizado para comunicação entre o serial gateway e controlador consiste de mensagens textuais em que cada dado é separado por ponto e virgula (;) e quebra de linha no final da mensagem. Sendo assim a mensagem possui a seguinte estrutura:

node-id; child-sensor-id; message-type; ack; sub-type; payload;

node-id identificação exclusiva do nó que envia ou deve receber a mensagem (endereço);

child-sensor-id Cada nó pode ter varios sensores ligados, esse campo identifica qual é sensor “child” do nó;

message-type Tipo de mensagem enviada.

ack Outgoing: 0 = mensagem não reconhecida, 1 = pedido ack do nó de destino; Incoming: 0 = mensagem normal, 1 = esta é uma mensagem de ack;

sub-type Dependendo messageType este campo tem um significado diferente.

payload Carga útil (max 25 bytes).

3.4 Códigos Mysensors

Nesta seção é apresentado um exemplo de código da biblioteca MySensors. Esse código apresenta as principais métodos utilizados na maioria dos exemplos.

Listing 3.1: Código Mysensors

```
1  #include <MySensor.h>
2  #include <SPI.h>
3  #define ID 0
4  #define OPEN 1
5  #define CLOSE 0
6  MySensor gw;
7  MyMessage msg(ID, V_TRIPPED);
8  void setup() {
9    gw.begin();
10   gw.present(ID, S_DOOR);
11 }
12 void loop() {
13       gw.send(msg.set(OPEN));
14       delay(10000);
15 }
```

Para iniciar precisamos criar uma instancia da biblioteca MySensors e depois ativada-lá com a instrução gw.begin(). Na primeira vez que o nó é ligado, o controlador atribui um id único a ele. Esse id é salvo na memória EEPROM do Arduino. Caso ele seja religado ou resetado, o id é automaticamente resgatado. O sensor deve ser apresentado para o controlador. Para isso é utilizado a função gw.present(child-sensor-id, sensor-type).

Para enviar uma mensagem deve-se criar um container MyMessage usando a função msg(child-sensor-id, variable-type). No escopo da função loop a mensagem é enviada com o método send(msg.set(payload)).

3.5 Controlador Pimatic

Para as atividades desenvolvidas nesse material utilizamos o controlador Pimatic.

Pimatic [1] é um framework de automação residencial que é executado no Node.js. Ele fornece uma plataforma extensível comum para controle de casa e tarefas de automação [**?**].

A configuração do Pimatic é realizada por meio de somente um arquivo, denominado config.json. Esse arquivo está no formato JSON e é dividido em quatro seções: Configurações (settings), Plugins, Dispositivos (devices) e Regras (Rules). A seguir, exemplos das quatro seções são apresentados.

[1] http://www.mysensors.org/controller/pimatic

Capítulo 4

Projeto A

Quando começamos aprender uma nova linguagem, geralmente iniciamos com o exemplo mais básico, o *"Hello Word"*. Nesse primeiro projeto também vamos começar com a construção do *"Hello Word"* da biblioteca MySensors e Pimatic, monitorando apenas a temperatura e umidade com um único nó sensor. Neste projeto vamos construir um nó sensor simples com o intuito de monitorar a temperatura e umidade de uma sala.

4.1 Materiais

Para esse projeto vamos o utilizar os seguintes materiais:

- 2 Arduinos;
- Sensor de temperatura e umidade DTH11;
- 2 Rádios RF;
- Jumpers.

Para obter mais informações sobre os materiais consulte o capitulo 2.

4.2 Implementação

Para a implementação desta atividade é necessário a construção dos seguintes nós.

4.2.1 Gateway

Esquemático

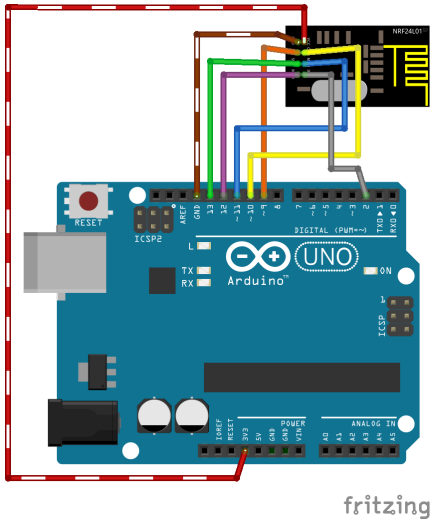

Figura 4.1: Gateway

Código

Listing 4.1: Gateway

```
1  #define NO_PORTB_PINCHANGES
2
3  #include <MySigningNone.h>
4  #include <MyTransportRFM69.h>
5  #include <MyTransportNRF24.h>
6  #include <MyHwATMega328.h>
7  #include <MySigningAtsha204Soft.h>
8  #include <MySigningAtsha204.h>
9
10 #include <SPI.h>
11 #include <MyParserSerial.h>
12 #include <MySensor.h>
13 #include <stdarg.h>
14 #include <PinChangeInt.h>
15 #include "GatewayUtil.h"
16
17 #define INCLUSION_MODE_TIME 1
18
19 #define INCLUSION_MODE_PIN  3
20
21 #define RADIO_ERROR_LED_PIN 4
22 #define RADIO_RX_LED_PIN    6
23 #define RADIO_TX_LED_PIN    5
24
25
26 MyTransportNRF24 transport(RF24_CE_PIN, RF24_CS_PIN,
       RF24_PA_LEVEL_GW);
27
28
29 MyHwATMega328 hw;
30
31
32 #ifdef WITH_LEDS_BLINKING
33 MySensor gw(transport, hw /*, signer*/, RADIO_RX_LED_PIN,
       RADIO_TX_LED_PIN, RADIO_ERROR_LED_PIN);
34 #else
35 MySensor gw(transport, hw /*, signer*/);
36 #endif
37
38 char inputString[MAX_RECEIVE_LENGTH] = "";
39 int inputPos = 0;
40 boolean commandComplete = false;
41
42 void parseAndSend(char *commandBuffer);
43
44 void output(const char *fmt, ... ) {
```

```
45      va_list args;
46      va_start (args, fmt );
47      vsnprintf_P(serialBuffer, MAX_SEND_LENGTH, fmt, args);
48      va_end (args);
49      Serial.print(serialBuffer);
50  }
51
52
53  void setup()
54  {
55    gw.begin(incomingMessage, 0, true, 0);
56
57    setupGateway(INCLUSION_MODE_PIN, INCLUSION_MODE_TIME,
          output);
58
59    PCintPort::attachInterrupt(pinInclusion,
          startInclusionInterrupt, RISING);
60
61    serial(PSTR("0;0;%d;0;%d;Gateway startup complete.\n"),
          C_INTERNAL, I_GATEWAY_READY);
62  }
63
64  void loop()
65  {
66    gw.process();
67
68    checkButtonTriggeredInclusion();
69    checkInclusionFinished();
70
71    if (commandComplete) {
72      parseAndSend(gw, inputString);
73      commandComplete = false;
74      inputPos = 0;
75    }
76  }
77
78  void serialEvent() {
79    while (Serial.available()) {
80      char inChar = (char)Serial.read();
81      if (inputPos<MAX_RECEIVE_LENGTH-1 && !commandComplete) {
82        if (inChar == '\n') {
83          inputString[inputPos] = 0;
84          commandComplete = true;
85        } else {
86          inputString[inputPos] = inChar;
87          inputPos++;
88        }
89      } else {
90          inputPos = 0;
91      }
92    }
93  }
```

4.2.2 Nó com sensor DTH11

Esquemático

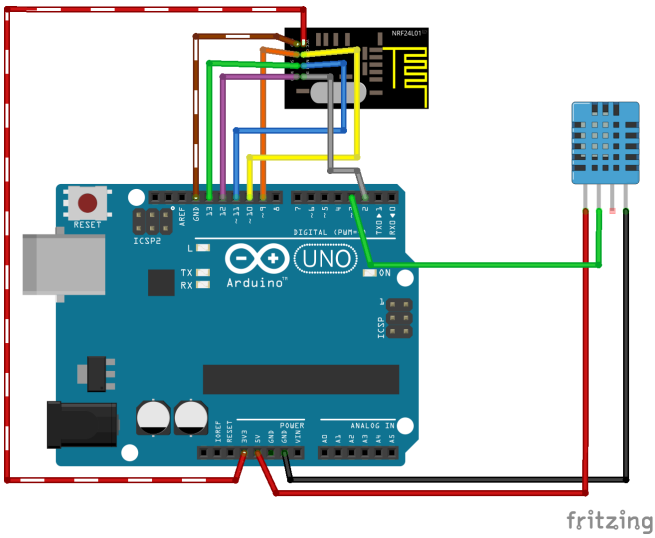

Figura 4.2: Sensor DTH11

Código

Listing 4.2: dth11

```
#include <SPI.h>
#include <MySensor.h>
#include <DHT.h>

#define CHILD_ID_HUM 0
#define CHILD_ID_TEMP 1
#define HUMIDITY_SENSOR_DIGITAL_PIN 3
unsigned long SLEEP_TIME = 30000;

MySensor gw;
DHT dht;
float lastTemp;
float lastHum;
boolean metric = true;
MyMessage msgHum(CHILD_ID_HUM, V_HUM);
MyMessage msgTemp(CHILD_ID_TEMP, V_TEMP);
int node_id = 2;

void setup()
{
  gw.begin(NULL, node_id);
  dht.setup(HUMIDITY_SENSOR_DIGITAL_PIN);


  gw.sendSketchInfo("Humidity", "1.0");


  gw.present(CHILD_ID_HUM, S_HUM);
  gw.present(CHILD_ID_TEMP, S_TEMP);

  metric = gw.getConfig().isMetric;
}

void loop()
{
  delay(dht.getMinimumSamplingPeriod());

  float temperature = dht.getTemperature();
  if (isnan(temperature)) {
      Serial.println("Failed reading temperature from DHT");
  } else if (temperature != lastTemp) {
    lastTemp = temperature;
    if (!metric) {
      temperature = dht.toFahrenheit(temperature);
    }
    gw.send(msgTemp.set(temperature, 1));
```

```
47      Serial.print("T: ");
48      Serial.println(temperature);
49    }
50
51    float humidity = dht.getHumidity();
52    if (isnan(humidity)) {
53        Serial.println("Failed reading humidity from DHT");
54    } else if (humidity != lastHum) {
55        lastHum = humidity;
56        gw.send(msgHum.set(humidity, 1));
57        Serial.print("H: ");
58        Serial.println(humidity);
59    }
60
61    gw.sleep(SLEEP_TIME);
62  }
```

4.2.3 Controlador

Arquivo de configuração do Pimatic

Listing 4.3: json.conf

```
1  {
2    "//": "Please only change this file when pimatic is NOT
         running, otherwise pimatic will overwrite your changes."
         ,
3    "settings": {
4      "httpServer": {
5        "enabled": true,
6        "port": 8080
7      },
8      "database": {
9      }
10   },
11   "plugins": [
12     {
13       "plugin": "cron"
14     },
15    {
16       "plugin": "mysensors",
17       "driver": "serialport",
18       "protocols": "1.4.1",
19       "driverOptions": {
20       "serialDevice": "/dev/ttyACM0",
21       "baudrate": 115200
22       }
23     },
24     {
```

```
25          "plugin": "mobile-frontend"
26        }
27      ],
28      "devices": [
29        {
30          "id": "DHT11_1",
31          "name": "Sensor Temperatura",
32          "class": "MySensorsDHT",
33          "nodeid": 2,
34          "sensorid": [
35            0,
36            1
37          ]
38        }
39
40      ],
41      "rules": [
42
43      ],
44      "pages": [
45        {
46          "id": "favourite",
47          "name": "Favourites",
48          "devices": []
49        }
50      ],
51      "groups": [
52
53      ],
54      "users": [
55        {
56          "username": "admin",
57          "password": "admin",
58          "role": "admin"
59        }
60      ],
61      "roles": [
62        {
63          "name": "admin",
64          "permissions": {
65            "pages": "write",
66            "rules": "write",
67            "variables": "write",
68            "messages": "write",
69            "events": "write",
70            "devices": "write",
71            "groups": "write",
72            "plugins": "write",
73            "updates": "write",
74            "database": "write",
75            "config": "write",
76            "controlDevices": true,
```

```
77            "restart": true
78          }
79        }
80    ]
81  }
```

Capítulo 5

Projeto B

Neste projeto construímos uma rede com dois nós sensores para monitorar a temperatura, umidade e luminosidade de um ambiente. Vamos construir um nó sensor para coletar informações de temperatura e umidade e outro nó para coletar informações de luminosidade do ambiente, os dois nós devem se comunicar com o controlador, o intuito desse projeto a apresentar como incluir mais nós em uma rede e como atualizar as informações do controlador para suportar os novos nós.

5.1 Materiais

Para esse projeto vamos o utilizar os seguintes materiais:

- 3 Arduinos;
- Sensor de temperatura e umidade DTH11;
- Sensor de luminosidade LDR;
- 3 Rádios RF;
- Jumpers.

Para obter mais informações sobre os materiais consulte o capitulo 2.

5.2 Implementação

5.2.1 Gateway

Esquemático

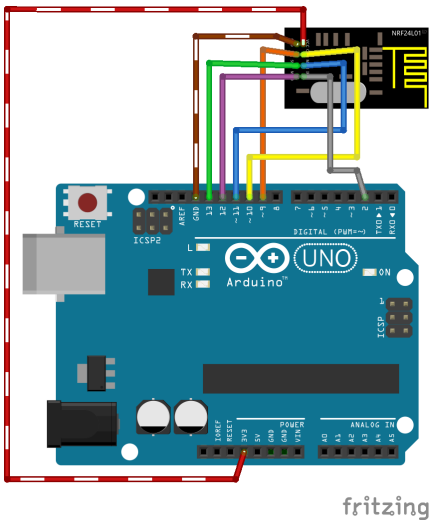

Figura 5.1: Gateway

Código

Listing 5.1: Gateway

```
1  #define NO_PORTB_PINCHANGES
2
3  #include <MySigningNone.h>
4  #include <MyTransportRFM69.h>
5  #include <MyTransportNRF24.h>
6  #include <MyHwATMega328.h>
7  #include <MySigningAtsha204Soft.h>
8  #include <MySigningAtsha204.h>
9
10 #include <SPI.h>
11 #include <MyParserSerial.h>
12 #include <MySensor.h>
13 #include <stdarg.h>
14 #include <PinChangeInt.h>
15 #include "GatewayUtil.h"
16
17 #define INCLUSION_MODE_TIME 1
18
19 #define INCLUSION_MODE_PIN  3
20
21 #define RADIO_ERROR_LED_PIN 4
22 #define RADIO_RX_LED_PIN    6
23 #define RADIO_TX_LED_PIN    5
24
25
26 MyTransportNRF24 transport(RF24_CE_PIN, RF24_CS_PIN,
       RF24_PA_LEVEL_GW);
27
28
29 MyHwATMega328 hw;
30
31
32 #ifdef WITH_LEDS_BLINKING
33 MySensor gw(transport, hw /*, signer*/, RADIO_RX_LED_PIN,
       RADIO_TX_LED_PIN, RADIO_ERROR_LED_PIN);
34 #else
35 MySensor gw(transport, hw /*, signer*/);
36 #endif
37
38 char inputString[MAX_RECEIVE_LENGTH] = "";
39 int inputPos = 0;
40 boolean commandComplete = false;
41
42 void parseAndSend(char *commandBuffer);
43
44 void output(const char *fmt, ... ) {
```

```
45      va_list args;
46      va_start (args, fmt );
47      vsnprintf_P(serialBuffer, MAX_SEND_LENGTH, fmt, args);
48      va_end (args);
49      Serial.print(serialBuffer);
50  }
51
52
53  void setup()
54  {
55    gw.begin(incomingMessage, 0, true, 0);
56
57    setupGateway(INCLUSION_MODE_PIN, INCLUSION_MODE_TIME,
          output);
58
59    PCintPort::attachInterrupt(pinInclusion,
          startInclusionInterrupt, RISING);
60
61    serial(PSTR("0;0;%d;0;%d;Gateway startup complete.\n"),
          C_INTERNAL, I_GATEWAY_READY);
62  }
63
64  void loop()
65  {
66    gw.process();
67
68    checkButtonTriggeredInclusion();
69    checkInclusionFinished();
70
71    if (commandComplete) {
72      parseAndSend(gw, inputString);
73      commandComplete = false;
74      inputPos = 0;
75    }
76  }
77
78  void serialEvent() {
79    while (Serial.available()) {
80      char inChar = (char)Serial.read();
81      if (inputPos<MAX_RECEIVE_LENGTH-1 && !commandComplete) {
82        if (inChar == '\n') {
83          inputString[inputPos] = 0;
84          commandComplete = true;
85        } else {
86          inputString[inputPos] = inChar;
87          inputPos++;
88        }
89      } else {
90          inputPos = 0;
91      }
92    }
93  }
```

5.2.2 Nó com sensor DTH11

Esquemático

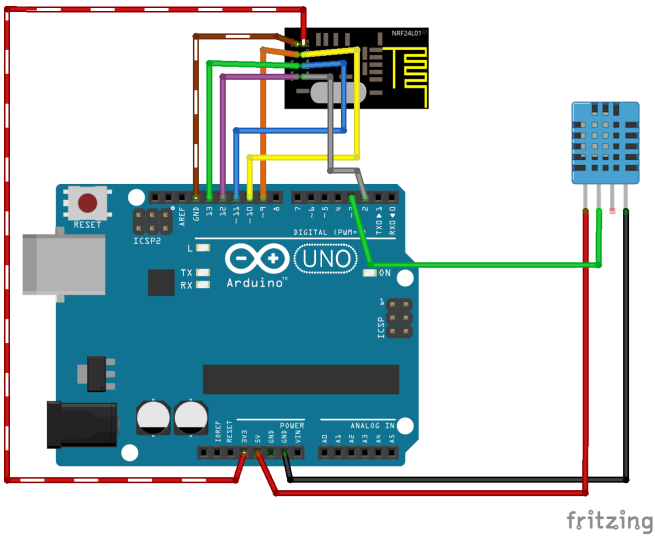

Figura 5.2: Sensor DTH11

Código

Listing 5.2: DTH11

```
#include <SPI.h>
#include <MySensor.h>
#include <DHT.h>

#define CHILD_ID_HUM 0
#define CHILD_ID_TEMP 1
#define HUMIDITY_SENSOR_DIGITAL_PIN 3
unsigned long SLEEP_TIME = 30000;

MySensor gw;
DHT dht;
float lastTemp;
float lastHum;
boolean metric = true;
MyMessage msgHum(CHILD_ID_HUM, V_HUM);
MyMessage msgTemp(CHILD_ID_TEMP, V_TEMP);
int node_id = 2;

void setup()
{
  gw.begin(NULL, node_id);
  dht.setup(HUMIDITY_SENSOR_DIGITAL_PIN);


  gw.sendSketchInfo("Humidity", "1.0");


  gw.present(CHILD_ID_HUM, S_HUM);
  gw.present(CHILD_ID_TEMP, S_TEMP);

  metric = gw.getConfig().isMetric;
}

void loop()
{
  delay(dht.getMinimumSamplingPeriod());

  float temperature = dht.getTemperature();
  if (isnan(temperature)) {
      Serial.println("Failed reading temperature from DHT");
  } else if (temperature != lastTemp) {
    lastTemp = temperature;
    if (!metric) {
      temperature = dht.toFahrenheit(temperature);
    }
    gw.send(msgTemp.set(temperature, 1));
```

```
47        Serial.print("T: ");
48        Serial.println(temperature);
49    }
50
51    float humidity = dht.getHumidity();
52    if (isnan(humidity)) {
53        Serial.println("Failed reading humidity from DHT");
54    } else if (humidity != lastHum) {
55        lastHum = humidity;
56        gw.send(msgHum.set(humidity, 1));
57        Serial.print("H: ");
58        Serial.println(humidity);
59    }
60
61    gw.sleep(SLEEP_TIME);
62 }
```

5.2.3 Nó com sensor LDR

Esquemático

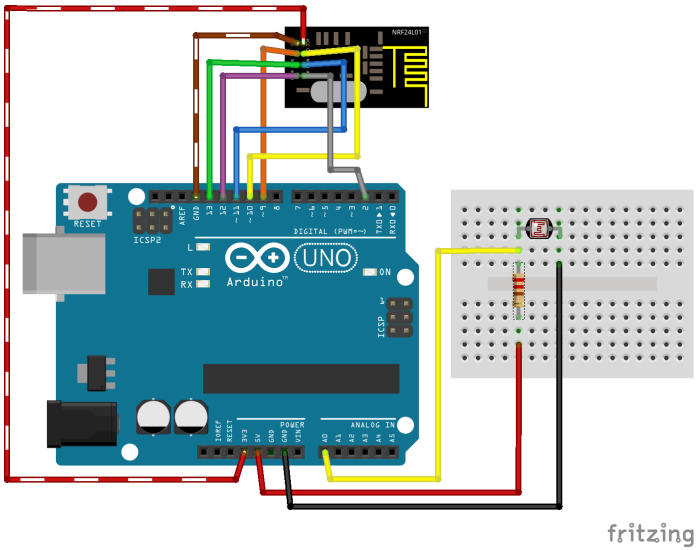

Figura 5.3: Sensor LDR

Código

Listing 5.3: LDR

```
#include <SPI.h>
#include <MySensor.h>

#define CHILD_ID_LIGHT 0
#define LIGHT_SENSOR_ANALOG_PIN 0

unsigned long SLEEP_TIME = 30000;

MySensor gw;
MyMessage msg(CHILD_ID_LIGHT, V_LIGHT_LEVEL);
int lastLightLevel;
int node_id = 3;

void setup()
{
  gw.begin(NULL, node_id);


  gw.sendSketchInfo("Light Sensor", "1.0");


  gw.present(CHILD_ID_LIGHT, S_LIGHT_LEVEL);
}

void loop()
{
  int lightLevel = (1023-analogRead(LIGHT_SENSOR_ANALOG_PIN))
      /10.23;
  Serial.println(lightLevel);
  if (lightLevel != lastLightLevel) {
      gw.send(msg.set(lightLevel));
      lastLightLevel = lightLevel;
  }
  gw.sleep(SLEEP_TIME);
}
```

5.2.4 Controlador

Arquivo de configuração do Pimatic

Listing 5.4: json.conf

```
{
  "//": "Please only change this file when pimatic is NOT
      running, otherwise pimatic will overwrite your changes."
      ,
  "settings": {
    "httpServer": {
      "enabled": true,
      "port": 8080
    },
    "database": {
    }
  },
  "plugins": [
    {
      "plugin": "cron"
    },
   {
      "plugin": "mysensors",
      "driver": "serialport",
      "protocols": "1.4.1",
      "driverOptions": {
      "serialDevice": "/dev/ttyACM0",
      "baudrate": 115200
      }
    },
    {
      "plugin": "mobile-frontend"
    }
  ],
  "devices": [
    {
      "id": "Light_1",
      "name": "Sensor LDR",
      "class": "MySensorsLight",
      "nodeid": 3,
      "sensorid": 1
    },
    {
      "id": "DHT11_1",
      "name": "Sensor Temperatura",
      "class": "MySensorsDHT",
      "nodeid": 2,
      "sensorid": [
        0,
        1
      ]
    }

  ],
  "rules": [

  ],
```

```
51    "pages": [
52      {
53        "id": "favourite",
54        "name": "Favourites",
55        "devices": []
56      }
57    ],
58    "groups": [
59
60    ],
61    "users": [
62      {
63        "username": "admin",
64        "password": "admin",
65        "role": "admin"
66      }
67    ],
68    "roles": [
69      {
70        "name": "admin",
71        "permissions": {
72          "pages": "write",
73          "rules": "write",
74          "variables": "write",
75          "messages": "write",
76          "events": "write",
77          "devices": "write",
78          "groups": "write",
79          "plugins": "write",
80          "updates": "write",
81          "database": "write",
82          "config": "write",
83          "controlDevices": true,
84          "restart": true
85        }
86      }
87    ]
88  }
```

Capítulo 6

Projeto C

Neste projeto construímos uma rede com dois nós. O primeiro nó contem dois sensores e o segundo nó contem um LED atuador. O nó contendo os sensores apresenta dois sensores um DTH11 para coletar informações de temperatura e umidade e um sensor LDR para monitor a luminosidade do ambiente. Este projeto possui o intuito de apresentar ao leitor o funcionamento de um nó com mais de um sensor e apresentar também um nó atuador. Vamos construir um nó sensor para coletar informações de temperatura, umidade e luminosidade e outro nó para emitir um sinal de alerta dependendo das informações dos sensores, os dois nós devem se comunicar com o controlador.

6.1 Materiais

Para esse projeto vamos o utilizar os seguintes materiais:

- 3 Arduinos;
- Sensor de temperatura e umidade DTH11;

- Sensor de luminosidade LDR;
- LED;
- Resistor 220 ohms;
- 3 Rádios nRF24L01;
- Jumpers;
- Protoboard.

Para obter mais informações sobre os materiais consulte o capitulo 2.

6.2 Implementação

6.2.1 Gateway

Esquemático

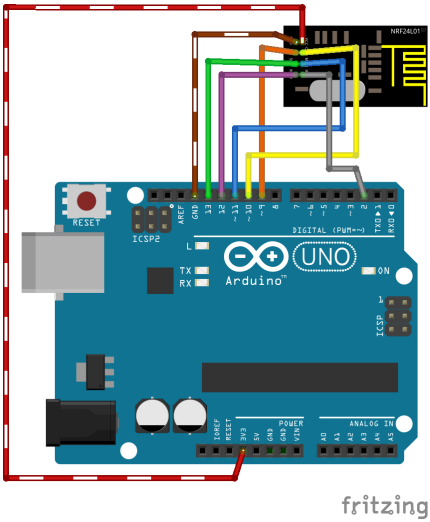

Figura 6.1: Gateway

Código

Listing 6.1: Gateway

```
1  #define NO_PORTB_PINCHANGES
2
3  #include <MySigningNone.h>
4  #include <MyTransportRFM69.h>
5  #include <MyTransportNRF24.h>
6  #include <MyHwATMega328.h>
7  #include <MySigningAtsha204Soft.h>
8  #include <MySigningAtsha204.h>
9
10 #include <SPI.h>
11 #include <MyParserSerial.h>
12 #include <MySensor.h>
13 #include <stdarg.h>
14 #include <PinChangeInt.h>
15 #include "GatewayUtil.h"
16
17 #define INCLUSION_MODE_TIME 1
18
19 #define INCLUSION_MODE_PIN  3
20
21 #define RADIO_ERROR_LED_PIN 4
```

```
22 #define RADIO_RX_LED_PIN      6
23 #define RADIO_TX_LED_PIN      5
24
25
26 MyTransportNRF24 transport(RF24_CE_PIN, RF24_CS_PIN,
       RF24_PA_LEVEL_GW);
27
28
29 MyHwATMega328 hw;
30
31
32 #ifdef WITH_LEDS_BLINKING
33 MySensor gw(transport, hw /*, signer*/, RADIO_RX_LED_PIN,
       RADIO_TX_LED_PIN, RADIO_ERROR_LED_PIN);
34 #else
35 MySensor gw(transport, hw /*, signer*/);
36 #endif
37
38 char inputString[MAX_RECEIVE_LENGTH] = "";
39 int inputPos = 0;
40 boolean commandComplete = false;
41
42 void parseAndSend(char *commandBuffer);
43
44 void output(const char *fmt, ... ) {
45    va_list args;
46    va_start (args, fmt );
47    vsnprintf_P(serialBuffer, MAX_SEND_LENGTH, fmt, args);
48    va_end (args);
49    Serial.print(serialBuffer);
50 }
51
52
53 void setup()
54 {
55   gw.begin(incomingMessage, 0, true, 0);
56
57   setupGateway(INCLUSION_MODE_PIN, INCLUSION_MODE_TIME,
         output);
58
59   PCintPort::attachInterrupt(pinInclusion,
         startInclusionInterrupt, RISING);
60
61   serial(PSTR("0;0;%d;0;%d;Gateway startup complete.\n"),
         C_INTERNAL, I_GATEWAY_READY);
62 }
63
64 void loop()
65 {
66   gw.process();
67
68   checkButtonTriggeredInclusion();
```

```
69    checkInclusionFinished();
70
71    if (commandComplete) {
72      parseAndSend(gw, inputString);
73      commandComplete = false;
74      inputPos = 0;
75    }
76  }
77
78  void serialEvent() {
79    while (Serial.available()) {
80      char inChar = (char)Serial.read();
81      if (inputPos<MAX_RECEIVE_LENGTH-1 && !commandComplete) {
82        if (inChar == '\n') {
83          inputString[inputPos] = 0;
84          commandComplete = true;
85        } else {
86          inputString[inputPos] = inChar;
87          inputPos++;
88        }
89      } else {
90          inputPos = 0;
91      }
92    }
93  }
```

6.2.2 Nó com sensor DTH11 e LDR

Esquemático

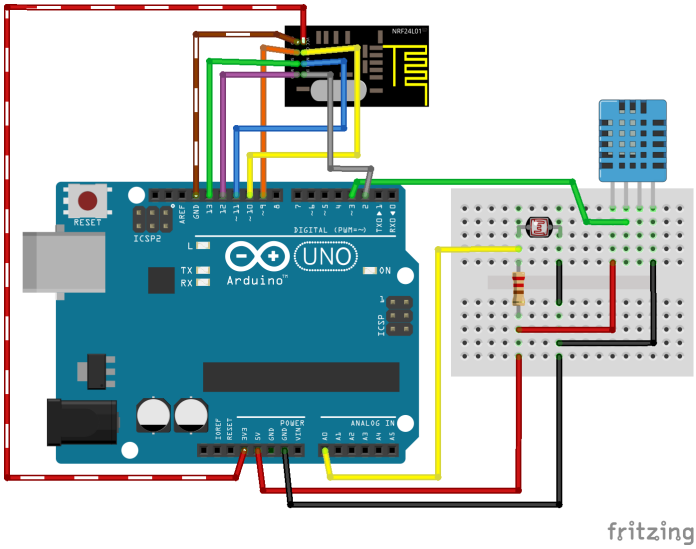

Figura 6.2: Nó Sensor DTH11 e LDR

Código

Listing 6.2: LDR e DTH11

```
1  #include <SPI.h>
2  #include <MySensor.h>
3  #include <DHT.h>
4
5  #define CHILD_ID_HUM 0
6  #define CHILD_ID_TEMP 1
7  #define HUMIDITY_SENSOR_DIGITAL_PIN 3
8
9  #define CHILD_ID_LIGHT 3
10 #define LIGHT_SENSOR_ANALOG_PIN 0
11
12 unsigned long SLEEP_TIME = 30000; // Sleep time between reads
       (in milliseconds)
13
14 MySensor gw;
15
16 MyMessage msg(CHILD_ID_LIGHT, V_LIGHT_LEVEL);
17 int lastLightLevel;
18
19 DHT dht;
```

```
20 float lastTemp;
21 float lastHum;
22 boolean metric = true;
23 MyMessage msgHum(CHILD_ID_HUM, V_HUM);
24 MyMessage msgTemp(CHILD_ID_TEMP, V_TEMP);
25 int node_id = 2;
26
27 void setup()
28 {
29   gw.begin(NULL, node_id);
30
31   gw.sendSketchInfo("Light Sensor", "1.0");
32   gw.present(CHILD_ID_LIGHT, S_LIGHT_LEVEL);
33
34   dht.setup(HUMIDITY_SENSOR_DIGITAL_PIN);
35
36   // Send the Sketch Version Information to the Gateway
37   gw.sendSketchInfo("Humidity", "1.0");
38
39   // Register all sensors to gw (they will be created as
       child devices)
40   gw.present(CHILD_ID_HUM, S_HUM);
41   gw.present(CHILD_ID_TEMP, S_TEMP);
42
43   metric = gw.getConfig().isMetric;
44
45
46 }
47
48 void loop()
49 {
50
51   int lightLevel = (1023-analogRead(LIGHT_SENSOR_ANALOG_PIN))
       /10.23;
52   Serial.println(lightLevel);
53   if (lightLevel != lastLightLevel) {
54       gw.send(msg.set(lightLevel));
55       lastLightLevel = lightLevel;
56   }
57
58   delay(dht.getMinimumSamplingPeriod());
59
60   float temperature = dht.getTemperature();
61   if (isnan(temperature)) {
62       Serial.println("Failed reading temperature from DHT");
63   } else if (temperature != lastTemp) {
64     lastTemp = temperature;
65     if (!metric) {
66       temperature = dht.toFahrenheit(temperature);
67     }
68     gw.send(msgTemp.set(temperature, 1));
69     Serial.print("T: ");
```

```
70        Serial.println(temperature);
71    }
72
73    float humidity = dht.getHumidity();
74    if (isnan(humidity)) {
75        Serial.println("Failed reading humidity from DHT");
76    } else if (humidity != lastHum) {
77        lastHum = humidity;
78        gw.send(msgHum.set(humidity, 1));
79        Serial.print("H: ");
80        Serial.println(humidity);
81    }
82
83    gw.sleep(SLEEP_TIME); //sleep a bit
84 }
```

6.2.3 Nó com atuador LED

Esquemático

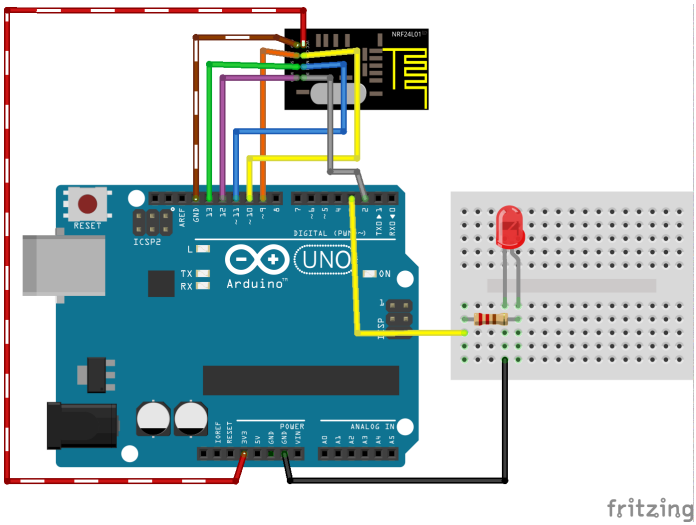

Figura 6.3: Nó LED

Código

Listing 6.3: LED

```
1  #include <MySigningNone.h>
2  #include <MyTransportNRF24.h>
3  #include <MyTransportRFM69.h>
4  #include <MyHwATMega328.h>
5  #include <MySensor.h>
6  #include <SPI.h>
7
8  #define RELAY_1  3
9  #define NUMBER_OF_RELAYS 1
10 #define RELAY_ON 1
11 #define RELAY_OFF 0
12
13
14 MyTransportNRF24 radio(RF24_CE_PIN, RF24_CS_PIN,
       RF24_PA_LEVEL_GW);
15
16 MyHwATMega328 hw;
17
18 MySensor gw(radio, hw);
19 int node_id = 3;
20
21 void setup()
22 {
23
24   gw.begin(incomingMessage, node_id, true);
25
26   gw.sendSketchInfo("Relay", "1.0");
27
28
29   for (int sensor=1, pin=RELAY_1; sensor<=NUMBER_OF_RELAYS;
         sensor++, pin++) {
30
31     gw.present(sensor, S_LIGHT);
32
33     pinMode(pin, OUTPUT);
34
35     digitalWrite(pin, gw.loadState(sensor)?RELAY_ON:RELAY_OFF
           );
36   }
37 }
38
39
40 void loop()
41 {
42   // Alway process incoming messages whenever possible
43   gw.process();
```

```
44 }
45
46 void incomingMessage(const MyMessage &message) {
47   // We only expect one type of message from controller. But
         we better check anyway.
48   if (message.type==V_LIGHT) {
49      // Change relay state
50      digitalWrite(message.sensor-1+RELAY_1, message.getBool()
           ?RELAY_ON:RELAY_OFF);
51      // Store state in eeprom
52      gw.saveState(message.sensor, message.getBool());
53      // Write some debug info
54      Serial.print("Incoming change for sensor:");
55      Serial.print(message.sensor);
56      Serial.print(", New status: ");
57      Serial.println(message.getBool());
58    }
59 }
```

6.2.4 Controlador

Arquivo de configuração do Pimatic

Listing 6.4: json.conf

```
1  {
2    "//": "Please only change this file when pimatic is NOT
         running, otherwise pimatic will overwrite your changes."
         ,
3    "settings": {
4      "httpServer": {
5        "enabled": true,
6        "port": 8080
7      },
8      "database": {
9      }
10   },
11   "plugins": [
12     {
13       "plugin": "cron"
14     },
15     {
16       "plugin": "mysensors",
17       "driver": "serialport",
18       "protocols": "1.4.1",
19       "driverOptions": {
20       "serialDevice": "/dev/ttyACM0",
21       "baudrate": 115200
22       }
```

```
23        },
24        {
25          "plugin": "mobile-frontend"
26        }
27      ],
28      "devices": [
29        {
30          "id": "Light_1",
31          "name": "Sensor LDR",
32          "class": "MySensorsLight",
33          "nodeid": 2,
34          "sensorid": 3
35        },
36        {
37          "id": "Switch_LED",
38          "name": "LED",
39          "class": "MySensorsSwitch",
40          "nodeid": 3,
41          "sensorid": 1
42        },
43        {
44          "id": "DHT11_1",
45          "name": "Sensor Temperatura 01",
46          "class": "MySensorsDHT",
47          "nodeid": 2,
48          "sensorid": [
49            0,
50            1
51          ]
52        }
53
54      ],
55      "rules": [
56
57      ],
58      "pages": [
59        {
60          "id": "favourite",
61          "name": "Favourites",
62          "devices": []
63        }
64      ],
65      "groups": [
66
67      ],
68      "users": [
69        {
70          "username": "admin",
71          "password": "admin",
72          "role": "admin"
73        }
74      ],
```

```
75    "roles": [
76      {
77        "name": "admin",
78        "permissions": {
79          "pages": "write",
80          "rules": "write",
81          "variables": "write",
82          "messages": "write",
83          "events": "write",
84          "devices": "write",
85          "groups": "write",
86          "plugins": "write",
87          "updates": "write",
88          "database": "write",
89          "config": "write",
90          "controlDevices": true,
91          "restart": true
92        }
93      }
94    ]
95  }
```