

UNIVERSIDADE DO OESTE DE SANTA CATARINA - UNOESC  
ÁREA DE CIÊNCIAS EXATAS E DA TERRA

BRUNO ALVES DOS SANTOS  
MARCOS FERNANDO KIWIATKOVSKI  
MARIHELLY SANTINI  
MICHEL ELIMAR ZARPELON  
PATRICK VENTURIN

DESENVOLVIMENTO DE UM PROJETO DE INTEGRAÇÃO *SOFTWARE-  
HARDWARE* PARA O MONITORAMENTO E CONTROLE DE UMA ESTAÇÃO  
METEOROLÓGICA

Joaçaba, SC

2014

BRUNO ALVES DOS SANTOS  
MARCOS FERNANDO KIWIATKOVSKI  
MARIHELLY SANTINI  
MICHEL ELIMAR ZARPELON  
PATRICK VENTURIN

DESENVOLVIMENTO DE UM PROJETO DE INTEGRAÇÃO *SOFTWARE-  
HARDWARE* PARA O MONITORAMENTO E CONTROLE DE UMA ESTAÇÃO  
METEOROLÓGICA

Trabalho da disciplina de Programação Móvel referente  
a avaliação A1/4 apresentado ao Curso Superior de  
Engenharia de Computação da Universidade do Oeste de  
Santa Catarina – Campus de Joaçaba.

Professor: Msc. Geovani Rodrigo Scolaro

Joaçaba, SC

2014

## RESUMO

Este trabalho demonstra o desenvolvimento de um protótipo de projeto de integração *software-hardware* para o controle e monitoramento de uma estação meteorológica, qual é capaz de realizar a coleta de dados referente a temperatura, luminosidade, umidade relativa do ar e a localização geográfica da estação meteorológica. O trabalho foi dividido em etapas. A primeira concentra-se na implementação do sistema embarcado, qual foi utilizado a placa de prototipação Arduino, e no desenvolvimento do *firmware*, qual realiza o controle da estação meteorológica. A segunda etapa trata no desenvolvimento do sistema de monitoramento, qual foi desenvolvido através de uma interface *web*, utilizando recursos da linguagem de programação Java e do *framework* JSF, e é responsável por requisitar valores atualizados de tempos em tempos à estação meteorológica e exibi-los na interface do sistema. Toda a comunicação foi implementada utilizando o protocolo de comunicação TCP/IP com a utilização de *sokets*. As etapas também seguiram a metodologia que é sugerida pela engenharia de *software* para o desenvolvimento de sistemas embarcados, que possibilitam uma visão detalhada da implementação do projeto quais contribuíram para o desenvolvimento do projeto.

Palavras-chave: Estação Meteorológica. Arduino. Sistemas Embarcados. Sistemas *Web*.

## LISTA DE ILUSTRAÇÕES

Esquema 1 – Visão geral do Projeto.....	6
Fotografia 1 – Estação Meteorológica em Hulha Negra (RS).....	10
Esquema 2 – Modelo de Arquitetura em Três Camadas. ....	10
Esquema 3 – Possível organização dos elementos de um Sistema Embarcado. ....	12
Diagrama 1 - Diagramas da Linguagem SysML. ....	17
Fotografia 2 - Placa Arduino Mega 2560 R3. ....	19
Fotografia 3 – Sensor DHT11 .....	21
Fotografia 4 - Sensor LDR. ....	21
Fotografia 5 - Módulo de comunicação GSM/GPRS SIM900.....	22
Fotografia 6 - Módulo GPS .....	23
Diagrama 2 – Escopo do Projeto. ....	23
Diagrama 3 - Requisitos Funcionais do <i>Software</i> Embarcado. ....	24
Diagrama 4 – Diagrama de Casos de Uso do <i>Software</i> Embarcado.....	25
Diagrama 5 – Requisitos Funcionais do Projeto de <i>Hardware</i> e <i>Software</i> Embarcado .	26
Esquema 4 - Conexões entre o sensor LM35 e a placa Arduino.....	27
Esquema 5 – Conexões entre o sensor LDR e a placa Arduino. ....	28
Esquema 6 - Conexão entre o sensor de umidade e a placa Arduino.....	28
Esquema 7 - Conexões entre o módulo GPS e a placa Arduino.....	29
Fotografia 7 – Encaixe do módulo GSM/GPRS SIM 900 e a placa Arduino.....	30
Fotografia 8 – Fluxo principal do Software Embarcado. ....	33
Fotografia 9 – Implementação da classe responsável pela coleta da Localização Geográfica. ....	35
Fotografia 10 – Implementação da classe responsável pela Comunicação. ....	36
Fotografia 11 – Implementação da classe responsável pela coleta da Temperatura. ....	37
Fotografia 12 – Implementação da classe responsável pela coleta da Luminosidade....	38
Fotografia 13 – Implementação da classe responsável pela coleta da Umidade. ....	38
Diagrama 6 – Escopo do Projeto. ....	38
Diagrama 7 – Levantamento dos Requisitos Funcionais. ....	39
Diagrama 8 – Diagrama de Casos de Uso da Estação de Monitoramento. ....	40
Fotografia 14 – Implementação da Classe Supervisor.....	42
Fotografia 15 – Sistema Embarcado montado na Estação Meteorológica .....	44
Fotografia 16 – Interface do Sistema <i>Web</i> .....	45
Fotografia 17 – Estação Meteorológica Montada .....	46
Fotografia 18 – Esquema para a Confecção do Anemômetro .....	49

## LISTA DE TABELAS

Tabela 1 – Especificações da placa Arduino. ....	19
Tabela 2 – Conversão para Obtenção das Coordenadas válidas ao Google <i>Maps</i> . ....	35

## SUMÁRIO

1	INTRODUÇÃO .....	7
1.1	APRESENTAÇÃO DO TEMA.....	8
1.2	OBJETIVOS .....	8
1.2.1	<b>Objetivos Gerais</b> .....	8
1.2.2	<b>Objetivos Específicos</b> .....	9
2	FUNDAMENTAÇÃO TEÓRICA.....	9
2.1	ESTAÇÃO METEOROLÓGICA .....	10
2.2	SISTEMAS <i>WEB</i> .....	11
2.3	SISTEMAS EMBARCADOS .....	12
2.4	ARDUINO .....	13
2.5	SISTEMA DE POSICIONAMENTO GLOBAL – GPS .....	14
2.5.1	<b>Sistema de Coordenadas e determinação da Posição Geográfica</b> .....	15
2.6	SERVIÇO GERAL DE RÁDIO COMUNICAÇÃO POR PACOTES – GPRS .....	16
2.7	MODELAGEM DE SISTEMAS EMBARCADOS .....	16
2.7.1	<b>SysML</b> .....	17
3	Metodologia .....	18
3.1.	COMPONENTES DE <i>HARDWARE</i> .....	18
3.2.	COMPONENTES DE <i>SOFTWARE</i> .....	23
3.3.	DESENVOLVIMENTO DO SISTEMA EMBARCADO .....	23
3.3.1	<b>Análise de Requisitos e Especificação</b> .....	24
3.3.2	<b>Arquitetura do Sistema e Projeto de Componentes</b> .....	26
3.3.3	<b>Integração dos Sistemas e Validações</b> .....	28
3.4	DESENVOLVIMENTO DO SISTEMA SUPERVISÓRIO .....	39
3.4.1	<b>Engenharia de Requisitos e Especificações</b> .....	39
3.4.2	<b>Integração dos dispositivos e sistemas</b> .....	41
5	<b>TRABALHOS RELACIONADOS</b> .....	43
6	RESULTADOS OBTIDOS .....	44
7	DISCUSSÕES .....	47
8	CONCLUSÃO .....	48
9	ANEXOS .....	49

## 1 INTRODUÇÃO

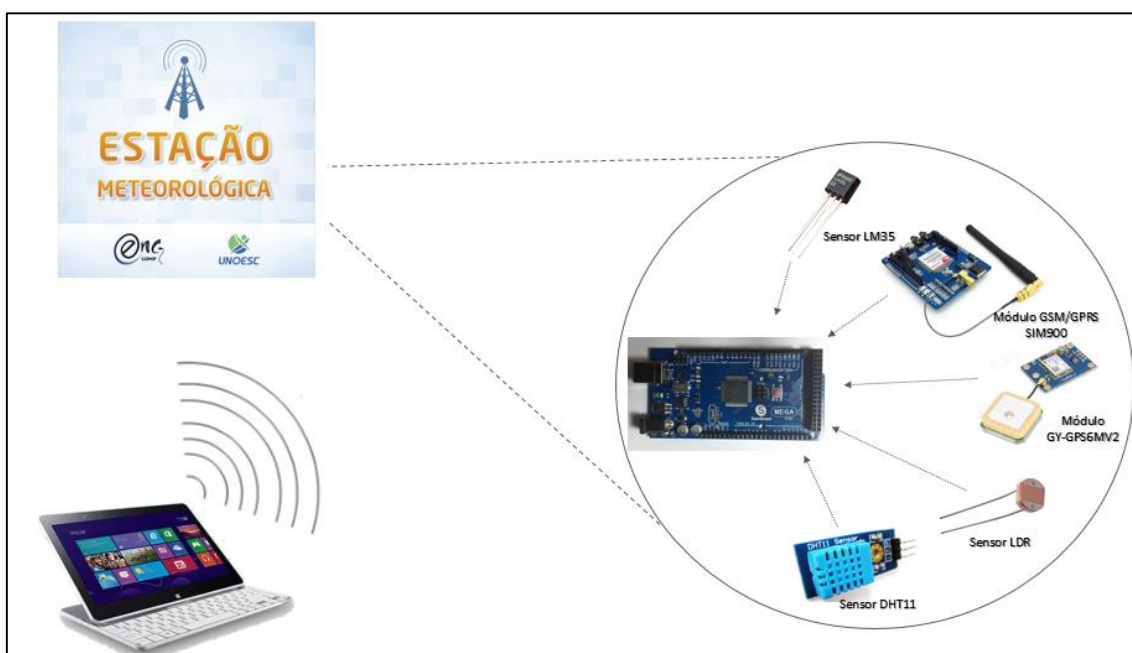
Uma estação meteorológica é um local onde são recolhidos dados para análise do tempo meteorológico. São equipadas com dispositivos e sensores capazes de realizar a captura dos dados e envia-los a uma estação de controle, e então são tratadas e exibidas em um sistema, em uma página de internet ou em um dispositivo móvel.

A aquisição dos dados meteorológicos é um dos primeiros itens que precisa ser desenvolvido. Para tanto, é necessário um *hardware* na estação meteorológica e um *software* para controla-lo, denominado *firmware*. Este conjunto forma um sistema embarcado e será desenvolvido utilizando uma placa de prototipagem para projetos eletrônicos, chamada Arduino.

A estação de monitoramento proposta neste projeto está em sistema *web*. Os dados exibidos neste sistema são atualizados em tempos em tempos, conforme as informações recebidas do sistema embarcado. Esta comunicação entre estação de monitoramento e o sistema embarcado localizado na estação meteorológica será realizada através de um módulo GPRS – *General Packet Radio Service* ou em português, Serviço Geral de Rádio Comunicação Por Pacotes. Sendo assim, é necessário apenas que a estação de monitoramento esteja localizada em uma área onde haja sinal de telefonia móvel.

No Esquema 1 pode-se observar a visão geral do projeto proposto onde é destacado o sistema embarcado da estação meteorológica.

Esquema 1 – Visão geral do Projeto.



Fonte: os autores.

## 1.1 APRESENTAÇÃO DO TEMA

Uma estação meteorológica é um local onde são recolhidos dados para análise do tempo meteorológico. Encontram-se equipadas com instrumentos (ou sensores eletrônicos) de medição e registro das variáveis meteorológicas/climáticas. Os seus dados são utilizados para a previsão do tempo e para a caracterização do clima, pelo que também podem ser designadas por estações climatológicas. Em nossos dias, por meio de programas de computador, integram-se os dados coletados, permitindo a sua apresentação. Na maior parte das estações de última geração os dados são enviados para computadores remotos, através de linhas telefônicas, redes GSM ou outros meios de transmissão.

Uma estação típica apresenta os seguintes instrumentos de medição:

- Termômetro para medir as variáveis da temperatura;
- Barômetro para medir as variáveis da pressão atmosférica;
- Higrômetro para medir as variáveis da umidade relativa do ar;
- Anemômetro para medir as variáveis da velocidade do vento;
- BirutaPB ou manga de ventoPE para indicar a orientação aproximada do vento;
- Pirômetro para medir as variáveis de insolação;
- Heliógrafo para medir a duração da ação do Sol;
- Pluviômetro para medir a quantidade de precipitação pluviométrica.

Outros dados passíveis de obtenção são o alcance visual de pista (visibilidade), altura de nuvens até aos 1500 metros, cobertura de céu nublado, nomeadamente para fins de navegação aérea.

## 1.2 OBJETIVOS

### 1.2.1 Objetivos Gerais

Este trabalho tem por objetivo desenvolver um projeto de integração *software-hardware* para o monitoramento de dados meteorológicos visando a implementação de um sistema embarcado, capaz de realizar a coleta de dados meteorológicos, e a



implementação de um sistema de monitoramento *web*, capaz de realizar a requisição de dados e monitoramento das mesmas.

### 1.2.2 Objetivos Específicos

Abaixo estão dispostos os objetivos específicos propostos por este projeto:

- Construção de um protótipo do sistema embarcado organizado da seguinte maneira:
  - Protótipo de *hardware* com a placa de prototipação Arduino, implementando a comunicação, através do módulo GPRS, e a transmissão de dados meteorológicos coletados: Luminosidade, temperatura, umidade relativa do ar e coordenadas de localização geográfica da estação meteorológica.
  - Protótipo de *software* embarcado (*firmware*) da estação meteorológica.
- Construção de um protótipo de *software* para a estação de monitoramento, para que seja realizado a requisição de dados à estação meteorológica e exibição das informações recebidas da mesma em um sistema *web*, composta por um mapa, indicando a localização geográfica da estação, e informações do tempo.
- Realização da integração entre o sistema embarcado e o *software* da estação de monitoramento;
- Realização da modelagem do sistema embarcado e da estação de monitoramento demonstrando aspectos e funcionalidades de comunicação e monitoramento da estação meteorológica.

## 2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são relacionados os assuntos que envolvem o projeto e seu desenvolvimento. Primeiramente é apresentado uma visão geral sobre o tema do projeto e em seguida é descrito definições de sistemas embarcados, placa de prototipação Arduino e sistemas *web*. Posteriormente, é apresentado os dispositivos e componentes de *hardware* utilizados para a comunicação e coleta de dados, que fizeram parte da implementação do projeto.

## 2.1 ESTAÇÃO METEOROLÓGICA

A aquisição de conhecimentos relativos ao tempo é um objetivo do ramo da ciência denominada meteorologia. Os fenômenos meteorológicos são estudados a partir das observações, experiências e métodos científicos de análise. A observação meteorológica é uma avaliação ou uma medida de um ou vários parâmetros meteorológicos. As observações são sensoriais quando são adquiridas por um observador sem ajuda de instrumentos de medição, e instrumentais, em geral chamadas medições meteorológicas, quando são realizadas com instrumentos meteorológicos.

Portanto, os instrumentos meteorológicos são equipamentos utilizados para adquirir dados meteorológicos (termômetro/temperatura do ar, pressão atmosférica/barômetro, higrômetro/umidade relativa do ar, etc.). A reunião desses instrumentos em um mesmo local, é denominada estação meteorológica.

Em nossos dias, por meio de programas de computador, integram-se os dados coletados, permitindo a sua apresentação. Na maior parte das estações de última geração os dados são enviados para computadores remotos, através de linhas telefônicas, redes GSM ou outros meios de transmissão.

No Brasil há o INMET – Instituto Nacional de Meteorologia e o INPE – Instituto Nacional de Pesquisas Espaciais, onde se fazem previsões que exigem maior precisão de dados. As informações sobre o tempo nas diversas regiões do Brasil, divulgadas pelos noticiários, são obtidas junto a esses institutos ou de outros similares (INMET, 2014).

Fotografia 1 – Estação Meteorológica em Hulha Negra (RS).



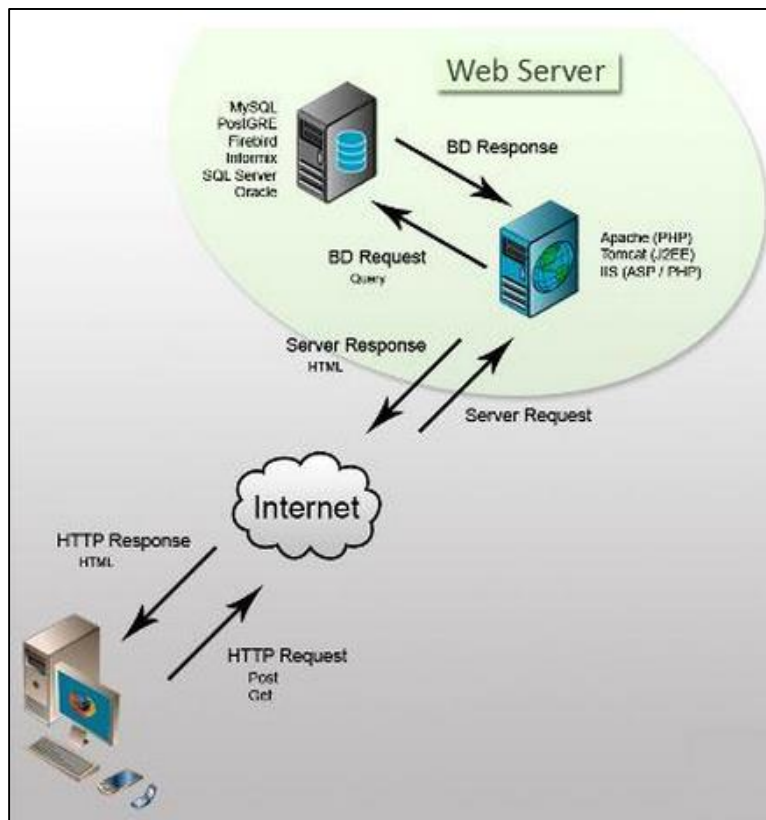
Fonte: Fernando Dias.

## 2.2 SISTEMAS WEB

Sistema *web* é o termo utilizado para designar, de forma geral, sistemas de informática projetados para utilização através de um navegador, na internet ou em redes privadas, como a Intranet. Um sistema *web* também é definido como tudo aquilo que se é processado em algum servidor com acesso à internet.

Uma arquitetura muito comum em aplicações *web* é o Modelo Arquitetural de Três Camadas. Nesse caso, temos uma camada de persistência (servidor de banco de dados), onde ficará os bancos de dados da aplicação. Temos também a camada de lógica de negócio (servidor de aplicação), onde rodará a aplicação *web* (seja ela Java, PHP, ASP ou qualquer outra linguagem) e uma camada de apresentação, que é representada pela máquina cliente que acessa a aplicação (FRANÇA, 2010).

Esquema 2 – Modelo de Arquitetura em Três Camadas.



Fonte: França (2010).

No nível de serviços, os elementos são organizados da seguinte forma: de um lado está o cliente *web*, ou *browser*, que solicita dados ao servidor *web*, recebe as respostas, formata a informação e a apresenta ao usuário. Do outro lado, está o servidor *web* que recebe as requisições, lê os dados (páginas HTML) do disco e as retorna para o cliente.

Esta seria a forma original do funcionamento *web*, onde é obtido somente páginas com conteúdo estático (apresentam sempre a mesma informação). Para que ocorra essa comunicação entre o software *web* e o usuário é necessária a utilização de páginas dinâmicas, onde o usuário faz a requisição via navegador e o servidor processa esta requisição devolvendo ao usuário uma nova resposta com uma página dinâmica HTML gerada (ARAÚJO, 1997).

Percebe-se que, a utilização de *softwares web* permitem ao usuário uma maior acessibilidade e leveza no carregamento das aplicações, abrindo assim a possibilidade de acesso aos sistemas e *sites* a partir de computadores *desktop* até dispositivos portáteis, tendo menores incompatibilidades aos sistemas operacionais existentes no mercado atualmente, devido ao ambiente de execução ser o navegador de internet. O desenvolvimento destes tipos de sistemas também facilita atualização e manutenção do *software* em questão, pois o código-fonte do mesmo fica armazenado em um único local (ARAÚJO, 1997).

## 2.3 SISTEMAS EMBARCADOS

Um sistema embarcado é um sistema micro processado e inserido em algum produto ou equipamento, com objetivos específicos, realizando tarefas predeterminadas (ASSEF, [2013?]).

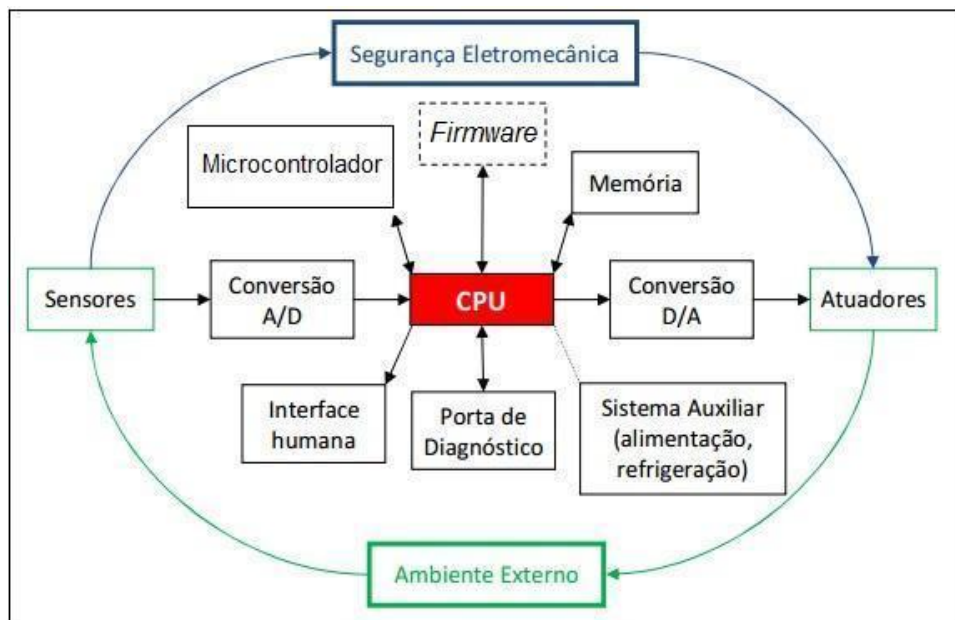
Também conhecidos como sistemas embutidos, são encontrados em diversos equipamentos utilizados no dia-a-dia, como *displays* gráficos, sistemas de monitoramento médico-hospitalar, câmeras fotográficas e de vídeo, eletrodomésticos, acionamento de motores, veículos de transporte etc. Com a popularização levou o desenvolvimento de programação com linguagens com maior poder de abstração, como a linguagem C, para a implementação das soluções embarcadas.

Os sistemas embarcados possuem *hardware* fixo, sem a possibilidade de alteração, porém podem ser adicionados a periféricos externos como *displays*, memórias, dispositivos de comunicação *wireless*, entre outros. Existem outra linha de sistemas embarcados que são compostos por *hardware* configurável, como os dispositivos lógicos programáveis CPLD e FPGA.

A comunicação entre um sistema embarcado e outros dispositivos é realizada através de interfaces de comunicação, como: UART, USB, RS232, RS485, I2C e SPI, Ethernet e WiFi.

O *software* desenvolvido para sistemas embarcados é muitas vezes chamado *firmware* ou também *software* embarcado. O mesmo é armazenado em uma memória do tipo ROM ou memória do tipo *flash* ao invés de um disco rígido (ASSEF, [2013?]).

Esquema 3 – Possível organização dos elementos de um Sistema Embarcado.



Fonte: adaptado de Machado (2011).

## 2.4 ARDUINO

Pode-se dizer que é uma placa de prototipagem ou uma plataforma de computação física ou embarcada, cujo funcionamento permite a interação com o ambiente através do uso de *software* e *hardware* (MCROBERTS, 2011).

O Arduino é uma plataforma de prototipagem eletrônica de *hardware* livre, projetada com um microcontrolador Atmel AVR de placa única, com suporte de entrada/saída embutido, uma linguagem de programação padrão, que é essencialmente C/C++. O objetivo do projeto é criar ferramentas que são acessíveis, com baixo custo, flexíveis e fáceis de se usar por artistas e amadores. Principalmente para aqueles que não teriam alcance aos controladores mais sofisticados e de ferramentas mais complicadas (ARDUINO, [2005?]).

Um *kit* básico de Arduino chega a custar 170 reais no mercado nacional, e conta com a placa controladora, *leds* de cores variadas, sensor de temperatura, cabo USB e outros itens diversos – o suficiente para iniciar um projeto básico. Outros componentes

podem ser adquiridos à parte, de acordo com a necessidade e disponibilidade financeira do usuário (VALLE, 2013).

Pode ser usado para o desenvolvimento de objetos interativos independentes, ou ainda para ser conectado a um computador hospedeiro. Uma típica placa Arduino é composta por um controlador, algumas linhas de E/S digital e analógica, além de uma interface serial ou USB, para interligar-se ao hospedeiro, que é usado para programá-la e interagir com o hospedeiro em tempo real. O Arduino também não se restringe a apenas uma placa principal, é possível adquirir outras placas, chamadas de *shields*, que contém uma variedade de outros dispositivos que podem ser conectadas à placa e utilizadas.

Ele contém tudo o necessário para suportar o microcontrolador, basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC para DC ou bateria para começar (ARDUINO, [2005?]).

## 2.5 SISTEMA DE POSICIONAMENTO GLOBAL – GPS

O Sistema de Posicionamento Global, conhecido por GPS (*Global Positioning System*), é um sistema de rádio navegação desenvolvido pelo Departamento de Defesa dos Estados Unidos, visando ser o principal sistema de navegação do exército americano. O GPS é um sistema de abrangência global, tal como o nome sugere. A concepção do sistema permite que um usuário em qualquer local da superfície terrestre tenha a sua disposição, no mínimo, quatro satélites que podem ser rastreados. Este número de satélite permite o posicionamento em tempo real (ALBUQUERQUE; SANTOS, 2003).

O Sistema de Posicionamento Global é composto por três componentes principais que operam em constante interação: o segmento espacial (satélites), o segmento de controle (monitoramento e controle terrestre) e o segmento do usuário (receptores GPS e equipamentos associados) (ALBUQUERQUE; SANTOS, 2003).

O segmento espacial é composto por um conjunto de pelo menos 24 satélites. No ano de 2008, o número de satélites GPS aumentou para 32, o que contribuiu para uma grande melhora na precisão do posicionamento. Os satélites são distribuídos em 6 planos orbitais, sendo que, esses planos têm uma inclinação de 55° em relação ao Equador. A configuração do segmento espacial foi projetada de forma que pelo menos 4 satélites estejam sempre acima do horizonte com uma probabilidade de 95%, nas 24 horas do dia em qualquer ponto da Terra.

Em razão da alta exatidão proporcionada pelo sistema e do alto grau de desenvolvimento da tecnologia envolvida nos receptores GPS, uma grande comunidade de usuários emergiu nas mais variadas aplicações civis, como navegação, posicionamento geodésico e topográfico, etc. As principais funções do GPS estão listadas abaixo:

- Posicionamento: dados de posicionamento tridimensional (latitude, longitude e altitude);
- Medição de velocidade: é possível estimar um tempo de chegada de acordo com a velocidade de um objeto;
- Medição de distância: os dados adquiridos através do GPS podem ser utilizados para a medição da distância entre dois pontos;

### 2.5.1 Sistema de Coordenadas e determinação da Posição Geográfica

O sistema de coordenadas utilizado pelo GPS é chamado de Sistema de Coordenadas Geodésico. Neste sistema a superfície de referência da Terra não é um plano horizontal, sendo representado por uma elipsoide (acompanhando a curvatura da Terra) visto que o nosso planeta é uma esfera ligeiramente achatada nos polos e mais larga na linha do equador. O sistema utiliza a latitude, longitude e altitude para representar a localização na superfície do planeta.

Latitude é a coordenada geográfica ou geodésica definida na esfera, que é o ângulo entre o plano do equador e a normal à superfície de referência. A latitude mede-se para norte e para sul do equador, entre 90° sul, no Polo Sul (negativa), e 90° norte, no Polo Norte (positiva). A latitude no equador é 0° (ALBUQUERQUE; SANTOS, 2003).

A Longitude descreve a localização de um lugar na Terra medido em graus, de zero a 180 para leste ou para oeste, a partir do Meridiano de Greenwich. Cada grau de longitude é subdividido em 60 minutos, e estes em 60 segundos. Uma longitude é especificada no formato graus° minutos' segundos". Caso a localidade esteja no oeste põe-se um sinal negativo (-) na frente da longitude. Ao invés de usar o sinal negativo, pode-se também usar as letras “E” e “W” para indicar "Leste" e "Oeste", respectivamente.

A altitude utiliza como referência o nível do mar. É a medida, no sentido vertical, entre o nível do mar e um ponto. A determinação da posição, através de um receptor de GPS, é feita através da medição das distâncias entre este receptor e os satélites (ALBUQUERQUE; SANTOS, 2003).

## 2.6 SERVIÇO GERAL DE RÁDIO COMUNICAÇÃO POR PACOTES – GPRS

O *General Packet Radio Services* – GPRS, ou em português, Serviço Geral de Rádio Comunicação por Pacotes, é uma tecnologia que tem como objetivo “possibilitar o tráfego de dados por pacotes para que a rede de telefonia celular possa ser integrada à internet. O sistema GSM com o GPRS integrado recebeu o nome de geração 2.5G, que foi uma evolução essencial nas telecomunicações” (SANTOS, 2008).

Esta tecnologia foi desenvolvida diante da necessidade que a telefonia celular teve em ser integrada à internet. Esta necessidade de integração surgiu durante a segunda geração de celulares (2G). Quando o GPRS foi integrado à rede, a geração recebeu o nome de 2.5G. A chegada do GPRS dividiu o tráfego em duas partes, voz e dados. O tráfego de voz continua sendo tratado da mesma forma como na geração 2G e o tráfego de dados passou a ser tratado pela nova arquitetura GPRS. Esta arquitetura utiliza-se de toda a estrutura já existente na rede GSM, modificando alguns elementos e inserindo novos (SVERZUT, 2005).

A rede GPRS se comunica por comutação de pacotes, onde a origem envia um pacote com o endereço de destino em seu cabeçalho pela rede, e então o pacote é transmitido pela rede na qual é responsável por determinar o melhor caminho até o destino, com a rede GSM. Os outros componentes da rede GSM, implementados na geração 2G, continuaram utilizando a comutação de circuitos (ou seja, a conexão entre duas entidades alocada de forma em estar sempre disponível, ou seja, de forma permanente) (SANTOS, 2008).

## 2.7 MODELAGEM DE SISTEMAS EMBARCADOS

Para lidar com projetos complexos, linguagens de alto nível de abstração têm sido utilizadas. Dentre elas, destaca-se a linguagem UML (*Unified Modeling Language*) considerada padrão para modelagem de software. A UML é uma linguagem genérica que pode ser usada para modelar diferentes tipos de sistemas. No entanto, a generalidade faz com que a UML não suporte a modelagem de alguns aspectos específicos em um determinado domínio de aplicação. Para atender esta demanda, a UML oferece mecanismos de extensão através da criação de perfis específicos (MARQUES, 2012).

O projeto de um sistema embarcado envolve etapas com diferentes níveis de abstração. As etapas de projeto compreendem:



- Análise de Requisitos: definição dos requisitos do sistema;
- Especificação: detalhamento das funcionalidades do sistema;
- Arquitetura do Sistema: detalhamento interno do sistema, bem como os componentes que compõem o sistema;
- Projeto de Componentes: projetar os componentes do sistema;
- Integração do Sistema: integração dos componentes desenvolvidos para o sistema e a validação dos mesmos.

Em cada nível de abstração durante as etapas do projeto de um sistema embarcado, é necessário analisar o projeto, para determinar as características do estado atual do mesmo, refinar o projeto adicionando detalhes e verificar o projeto, avaliando se os objetivos dos sistemas, tais como custo, velocidade serão alcançados (MARQUES, 2012).

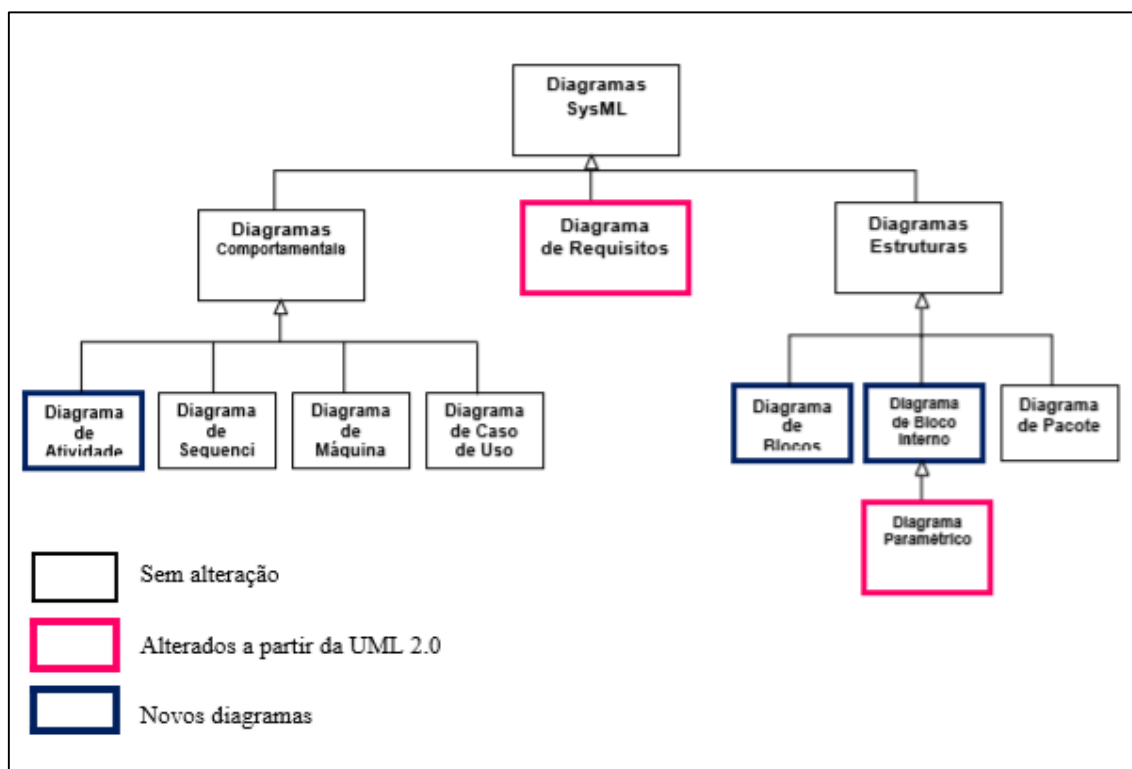
No domínio de sistemas embarcados, o suporte à descrição de comportamentos heterogêneos (*hardware/software* e diferentes modelos de computação) e a especificação de requisitos não-funcionais são exemplos de necessidades específicas do domínio. Dentre os perfis propostos pela OMG (*Object Management Group*), com enfoque no domínio de embarcados, destaca-se a linguagem SysML (*Systems Modeling Language*) (MARQUES, 2011).

### 2.7.1 SysML

A SysML é uma linguagem de modelagem gráfica de propósito geral que suporta especificação, análise, projeto, verificação e validação de sistemas complexos. Estes sistemas podem incluir *hardware*, *software*, informação, processos, pessoas e infraestrutura.

A SysML reutiliza um subconjunto da UML e adiciona extensões para atender os requisitos da UML SE RFP (*System Engineering Request for Proposal*) (MARQUES, 2011). O Diagrama 1 ilustra as modificações ocorridas nos diagramas reutilizados da UML 2.0, bem como os novos diagramas da SysML. Os diagramas são classificados em três classes: comportamentais, estruturais e de requisitos.

Diagrama 1 - Diagramas da Linguagem SysML.



Fonte: Marques (2012, apud OMG, 2010, p.180).

### 3 METODOLOGIA

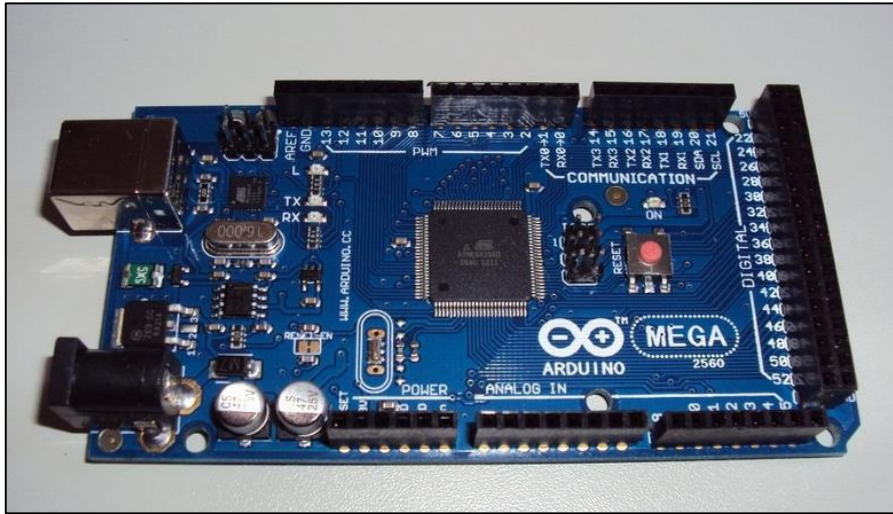
Nesta seção são apresentados e descritos os recursos, métodos e procedimentos adotados e como foram aplicados ao desenvolvimento do projeto.

#### 3.1 COMPONENTES DE *HARDWARE*

O projeto do hardware foi montado em uma matriz de contatos, denominada *proto-board*. Esta foi escolhida para o desenvolvimento do sistema embarcado devido a facilidade de alteração do circuito. A fotografia XX mostra uma matriz de contatos, similar a utilizada no projeto.

O desenvolvimento do projeto foi realizado utilizando a placa de prototipação Arduino Mega 2560 R3. Este, possui 256 KB de memória *flash* para armazenamento de código (dos quais 8 KB é usado para o *bootloader*), 8 KB de SRAM e 4 KB de EEPROM (ARDUINO, [2005?]).

Fotografia 2 - Placa Arduino Mega 2560 R3.



Fonte: os autores.

A placa pode ser alimentada conectando-a via cabo USB em um computador ou então conectando em uma fonte AC-DC ou bateria. A tabela abaixo descreve as especificações da placa detalhadamente.

Tabela 1 - Especificações da placa Arduino.

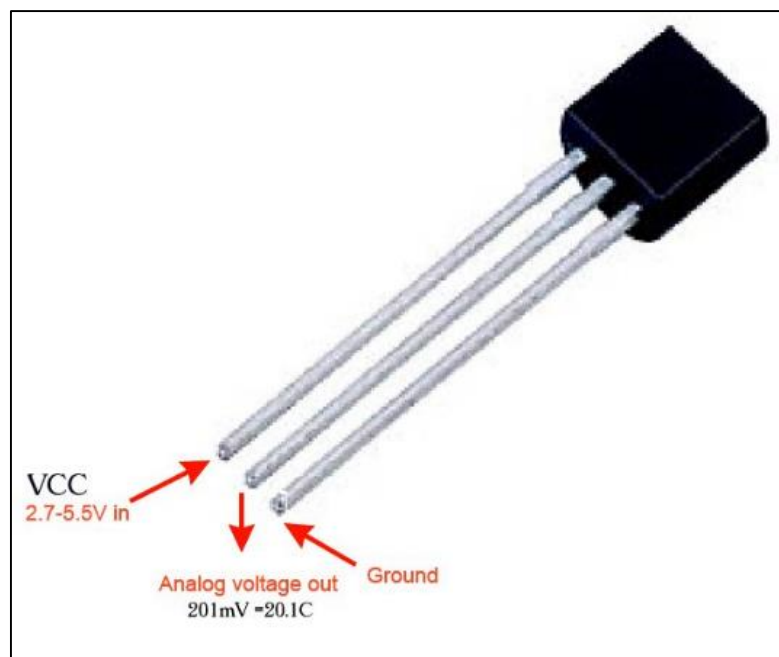
Característica	Descrição
Microcontrolador	ATmega2560
Tensão de funcionamento	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limites)	6-20V
Digital pinos de I/O	54 (dos quais 15 oferecem saída PWM)
Entrada Analógica Pinos	16
DC Current per I/O Pin	40 mA
Corrente DC para Pin 3.3V	50 mA
Memória Flash	256 KB dos quais 8 KB usados pelo <i>bootloader</i>
SRAM	8 KB
EEPROM	4 KB
Velocidade do <i>clock</i>	16 MHz

Fonte: Arduino ([2005?]).

Cada um dos pinos 54 digitais da placa pode ser usado como uma entrada ou uma saída, definindo os mesmos através das funções *pinMode()*, *digitalWrite()*, e *digitalRead()*, eles operam a 5 volts. Cada pino pode fornecer ou receber o máximo de 40 mA e tem um resistor *pull-up* interno (desconectado por padrão) de 20-50 *kOhms*. O mega tem 16 entradas analógicas, cada uma das quais com 10 *bits* de resolução (1024 valores diferentes). Por padrão, eles medem de terra para 5 volts (ARDUINO, [2005?]).

Para a coleta dos dados referente a temperatura, foi utilizado o sensor LM35, fabricado pela National Semiconductor. Ele apresenta uma saída de tensão linear relativa à temperatura em que ele se encontrar no momento em que for alimentado por uma tensão (4-20Vdc e GND). A saída é um sinal de 10mV para cada Grau Celsius de temperatura, não necessitando nenhuma subtração de variáveis para que se obtenha uma escala de temperatura em Graus Celsius (CRESPI; CERON, [2010?]).

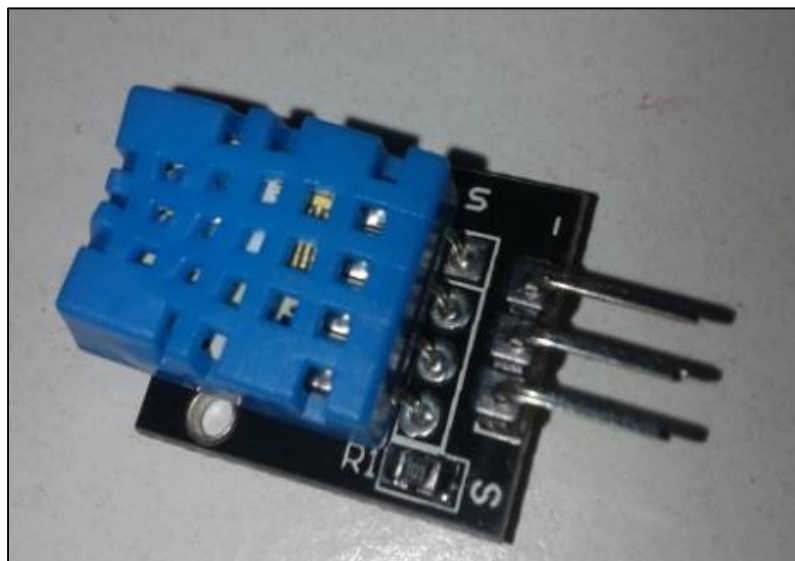
Figura 3 – Sensor de temperatura LM35.



Fonte: Prasad (2011).

Para a coleta da umidade, foi utilizado o DHT11 que é um sensor capaz de interpretar as variáveis físicas de umidade relativa e temperatura, com saída digital calibrada. Possui uma exclusiva tecnologia para medir a umidade, garantindo a confiabilidade e estabilidade. Todos os sensores desse modelo são calibrados de fábrica, e os dados e coeficientes da calibração estão gravados na memória OTP do módulo (D-Robotics UK, 2010).

Fotografia 3 – Sensor DHT11



Fonte: os autores.

O sensor apresenta o formato retangular e possui quatro pinos, onde apenas três deles são utilizados para coleta das informações de umidade e temperatura ambiente (D-Robotics UK, 2010).

Para a coleta da luminosidade, foi utilizado o sensor LDR (*Light Dependent Resistor*), em português Resistor Dependente de Luz. Este componente é um resistor cuja resistência varia conforme a intensidade da luz que incide sobre ele. Ele é construído a partir de material semicondutor com elevada resistência elétrica.

Fotografia 4 - Sensor LDR.



Fonte: os autores.

Quando a luz que incide sobre o semicondutor tem uma frequência suficiente, os fótons que incidem sobre o semicondutor libertam elétrons para a banda condutora que irão melhorar a sua condutividade e assim diminuir a resistência. Os resultados típicos para um LDR poderão ser:

- Escuridão: resistência máxima, geralmente mega *ohms*.
- Luz muito brilhante: resistência mínima, geralmente dezenas de *ohms*.

Dependendo do tipo, um LDR pode ser sensível às faixas de luz: Infravermelhas (IR), Luz visível ou Ultravioletas (UV).

A comunicação entre o sistema embarcado e a estação de monitoramento *web* foi realizada através do módulo GSM/GPRS SIM900, da fabricado pela IComsat, indicado na Fotografia 5. Este dispositivo é controlado através de Comandos AT. O mesmo também possui suporte a áudio, conta com microfones e saída para fones de ouvido.

Fotografia 5 - Módulo de comunicação GSM/GPRS SIM900.



Fonte: os autores.

O módulo GPS utilizado no projeto foi o GY-NEO6MV2 da fabricante UBlox. Este componente é bastante pequeno, possui 25 x 35mm de tamanho e possui uma antena de cerâmica, com tamanho de 25 x 25mm. A taxa de transmissão padrão é de 9600bps e

sua tensão de alimentação é varia de 3,3 a 4,3 volts. A Fotografia 6 demonstra o componente utilizado no projeto.

Fotografia 6 - Módulo GPS



Fonte: os autores.

### 3.2 COMPONENTES DE *SOFTWARE*

Para o desenvolvimento do *software* embarcado, foi utilizado o próprio ambiente de desenvolvimento do Arduino. É a linguagem de programação padrão do Arduino que foi utilizada, qual tem origem em *Wiring*, e é essencialmente C/C++.

Já para o desenvolvimento da estação de monitoramento *web* foi utilizado o ambiente de desenvolvimento do NetBeans. A linguagem utilizada foi Java, utilizando recursos do Java Server Faces (JSF) que é um *framework* MVC para a construção de interfaces de usuário baseadas em componentes para aplicações *web*.

### 3.3 DESENVOLVIMENTO DO SISTEMA EMBARCADO

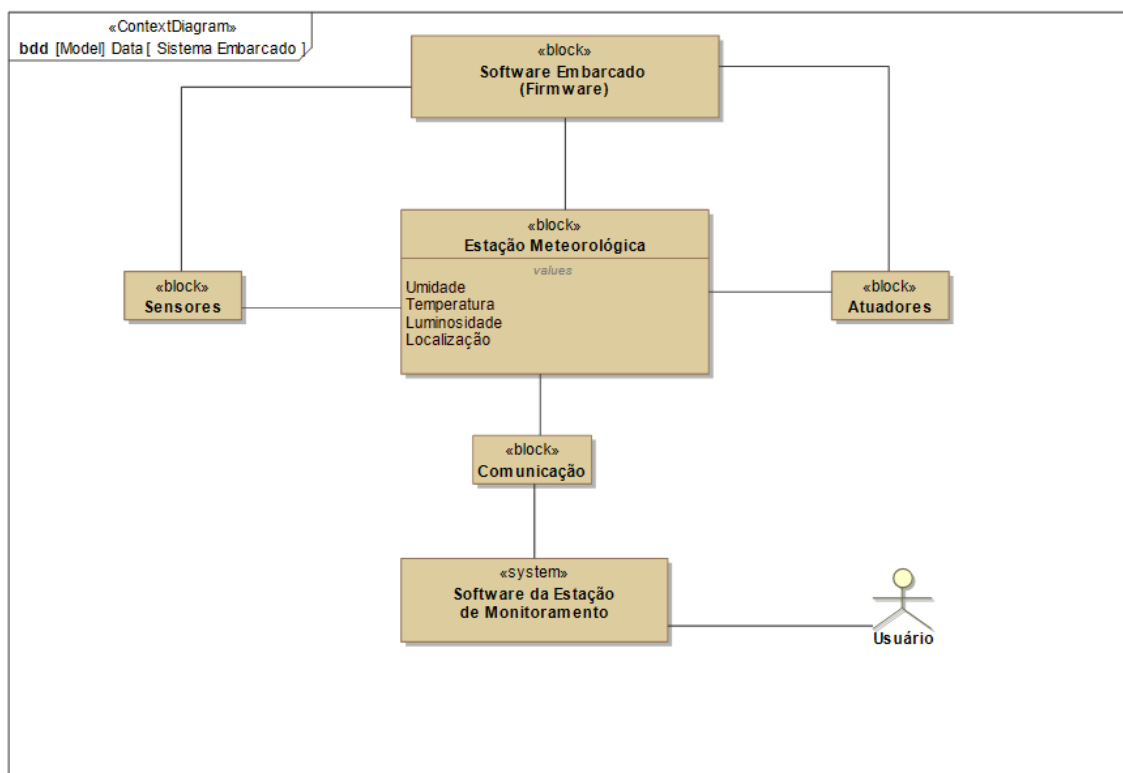
Nas seções abaixo serão apresentados os resultados do projeto referente ao desenvolvimento sistema embarcado, demonstrando como o mesmo foi projetado e instalados na estação meteorológica, seguindo a metodologia de desenvolvimento de sistemas embarcados propostas pela Engenharia de Software e que foram detalhadas na seção 2.7.

### 3.3.1 Análise de Requisitos e Especificação

Os diagramas da SysML foram utilizados como forma de documentação e para auxiliar na organização para o desenvolvimento do projeto do sistema embarcado.

O Diagrama 2 demonstra o escopo do projeto, o qual demonstra de uma forma clara uma visão generalizada do que haverá de ser desenvolvido. Este foi projetado utilizando o diagrama de contexto da SysML.

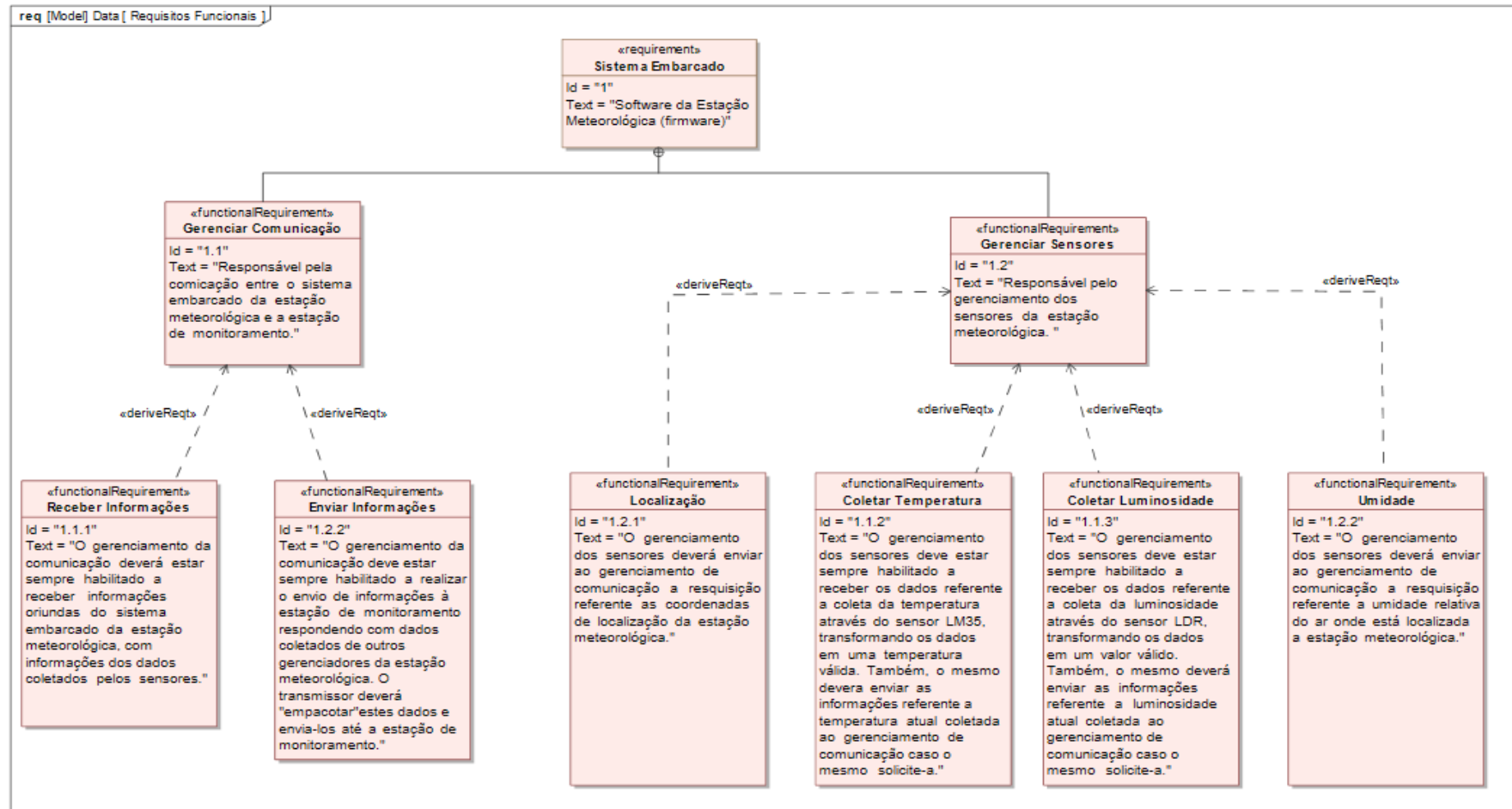
Diagrama 2 – Escopo do Projeto.



Fonte: os autores.

O levantamento de requisitos do *software* embarcado foi realizado com o auxílio do diagrama de requisitos da SysML. No diagrama 2 é possível visualizar os requisitos funcionais levantados para o *software* embarcado.

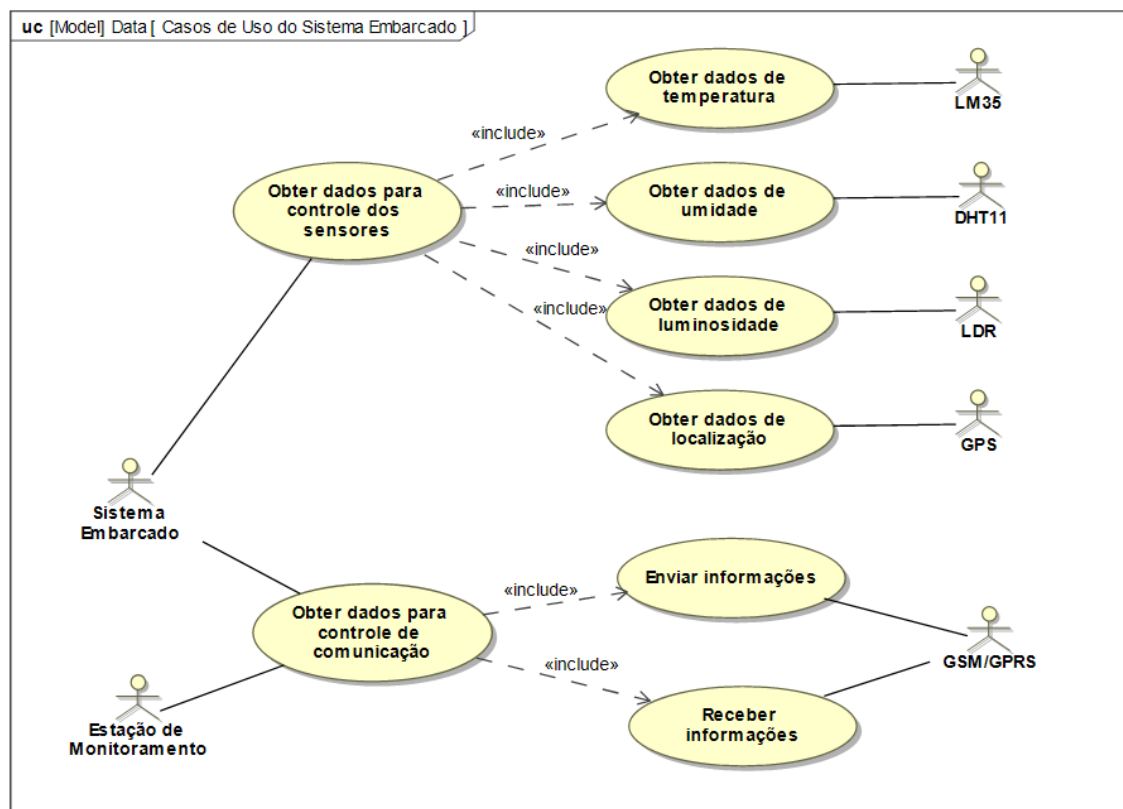


Diagrama 3 - Requisitos Funcionais do *Software* Embarcado.

Fonte: os autores.

Na etapa posterior foi organizado todos os requisitos levantados através da identificação dos casos de uso. O Diagrama 4 demonstra os casos de uso organizados, objetivando demonstrar o uso dos objetos no sistema.

Diagrama 4 – Diagrama de Casos de Uso do *Software* Embarcado.

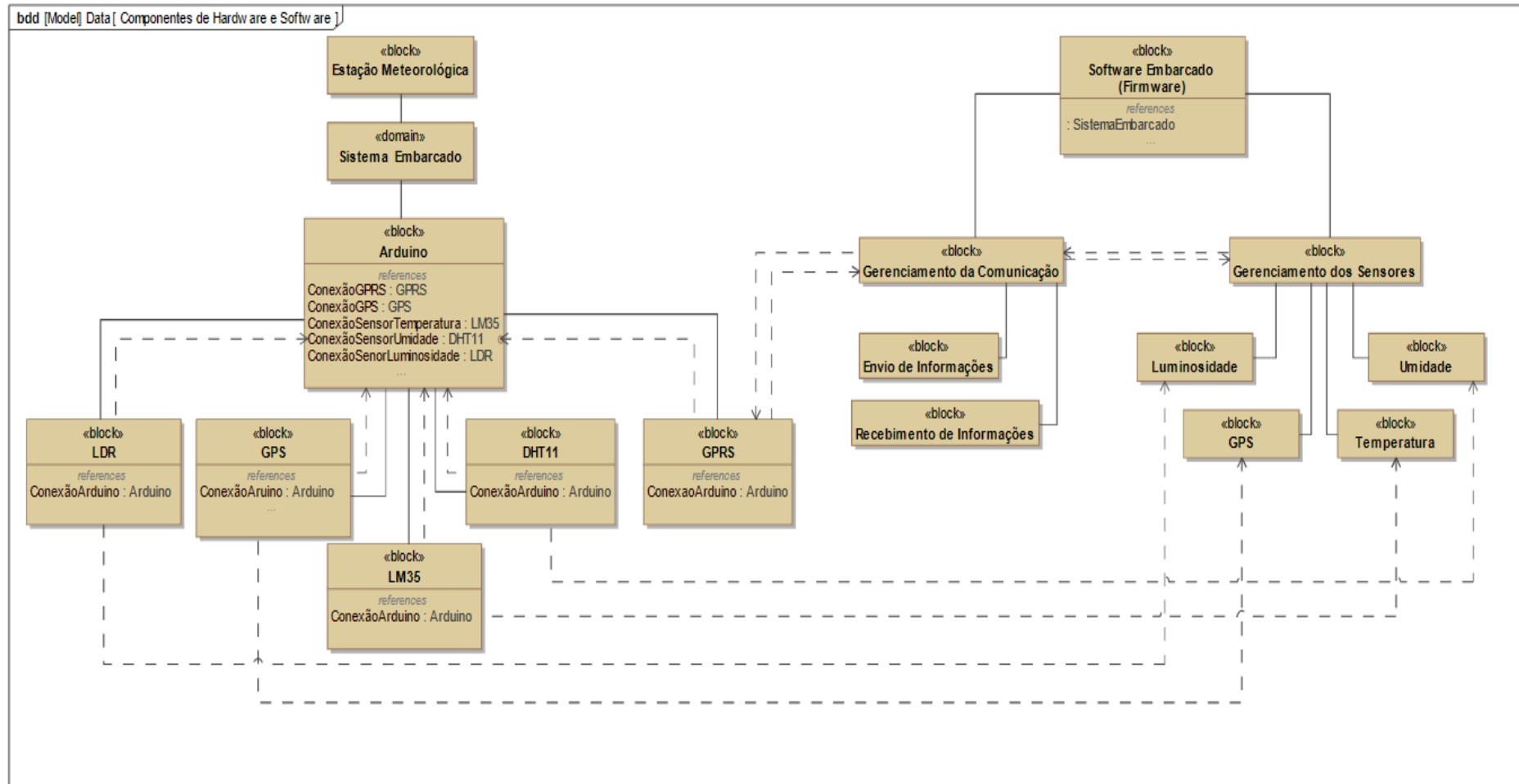


Fonte: os autores.

Seguindo as orientações da Engenharia de *Software*, ainda haveria a expansão dos casos de uso. Estes não foram desenvolvidos devido a limitação do tempo de desenvolvimento deste projeto.

### 3.3.2 Arquitetura do Sistema e Projeto de Componentes

Nesta etapa foi desenvolvida a modelagem da arquitetura do sistema embarcado. Para tanto foi utilizado o diagrama de blocos da SysML, no qual é possível modelar a arquitetura do sistema, representando os componentes de *hardware* e os componentes de *software* e a ligação entre os mesmos. O resultado da junção dos componentes de *hardware* e *software* da estação meteorológica são demonstrados no Diagrama 5.

Diagrama 5 - Requisitos Funcionais do Projeto de *Hardware* e *Software* Embarcado

Fonte: os autores.

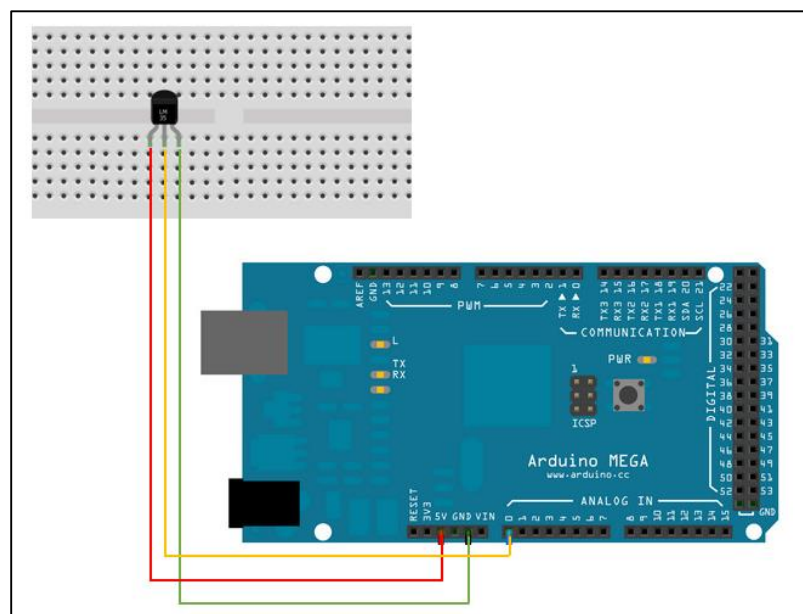
### 3.3.3 Integração dos Sistemas e Validações

Esta etapa do projeto consistiu em realizar a montagem do sistema embarcado, realizando a união de todos os dispositivos de *hardware*, quais já haviam sido levantados em seções anteriores. Em paralelo, foi realizado o desenvolvimento do *software* embarcado, que é o responsável pelo controle de todos os componentes da estação meteorológica. Segundo Marques (2012, apud Wolf, 2008) a fase de integração do sistema é um dos maiores desafios no projeto de sistemas embarcados, pois é nesta fase que os componentes que foram construídos são unificados e estes devem funcionar corretamente.

Todos os sensores e componentes de *hardware* foram montados separadamente, a fim de realizar os testes verificando se tudo estava funcionando de acordo, conforme o esperado. Para cada componente foi realizado a montagem de um esquema demonstrando como é realizada as ligações do dispositivo com a placa Arduino.

Foi realizada a conexão do sensor de temperatura LM35 na placa Arduino, sendo responsável pela coleta da temperatura. Este sensor utiliza as portas analógicas do Arduino e deve ser alimentado a 5 *volts*, quais são demonstradas no Esquema 4.

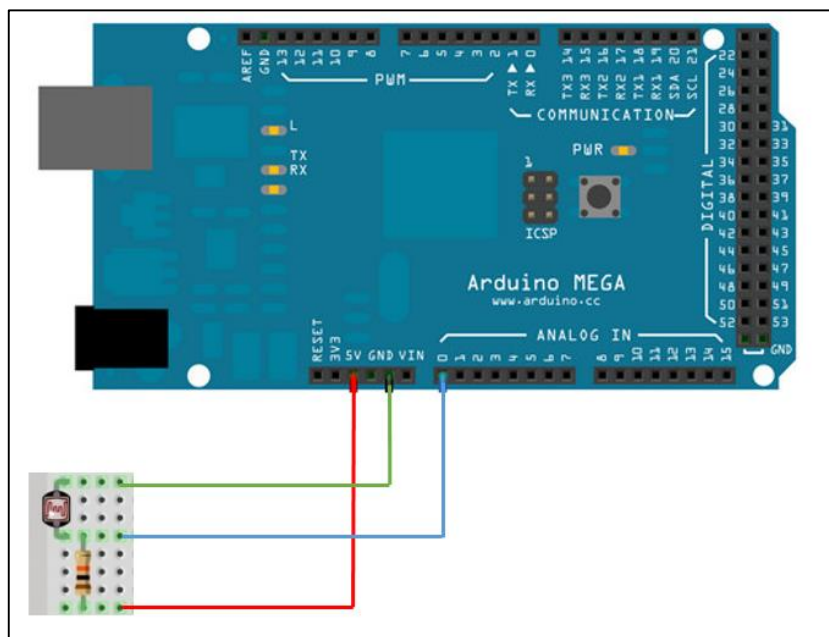
Esquema 4 - Conexões entre o sensor LM35 e a placa Arduino.



Fonte: os autores.

O sensor LDR, responsável pela coleta da luminosidade, também utiliza as portas analógicas da placa Arduino. Este sensor é alimentado por 5 *volts* de tensão, porém, é necessário a utilização de um resistor de 10k *ohms*. O esquema 5 demonstra suas ligações.

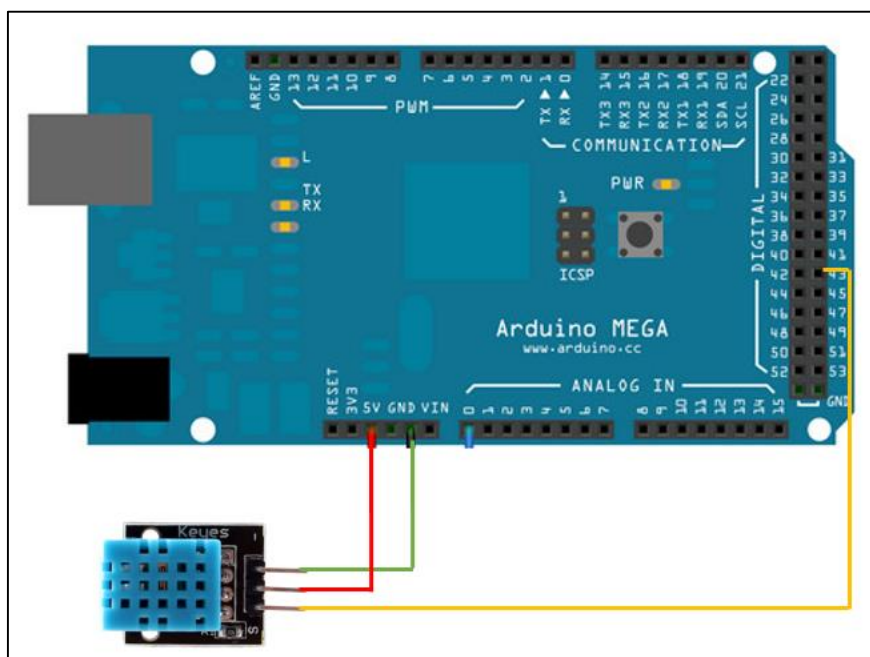
Esquema 5 – Conexões entre o sensor LDR e a placa Arduino.



Fonte: os autores.

O sensor DHT11, que é responsável pela coleta de umidade relativa do ar, utiliza as portas digitais da placa Arduino e é alimentado por uma tensão de 5 *volts*. Através dele também é possível obter a temperatura, mas, por questões de requisitos deste projeto, utilizamos o sensor LM35 para esta funcionalidade. O Esquema 6 demonstra suas ligações.

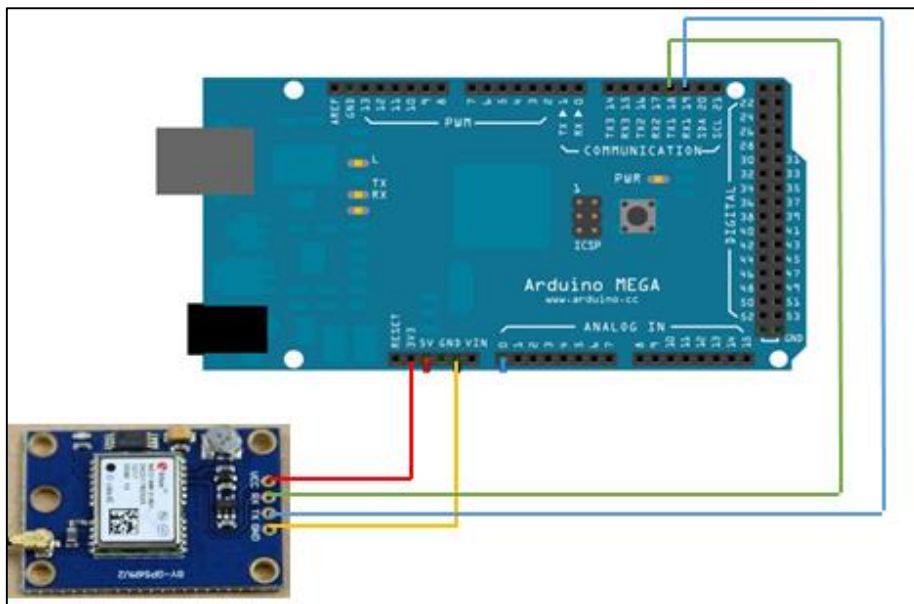
Esquema 6 - Conexão entre o sensor de umidade e a placa Arduino



Fonte: os autores.

O módulo GPS utiliza as portas seriais do Arduino. Este componente se comunica diretamente à placa utilizando os pinos de transmissão de dados RX e TX. Deve se tomar cuidado quanto a alimentação deste componente, pois a tensão máxima suportada pelo mesmo é de 3,3 *volts*. O esquema 7 demonstra as ligações entre o GPS e o Arduino.

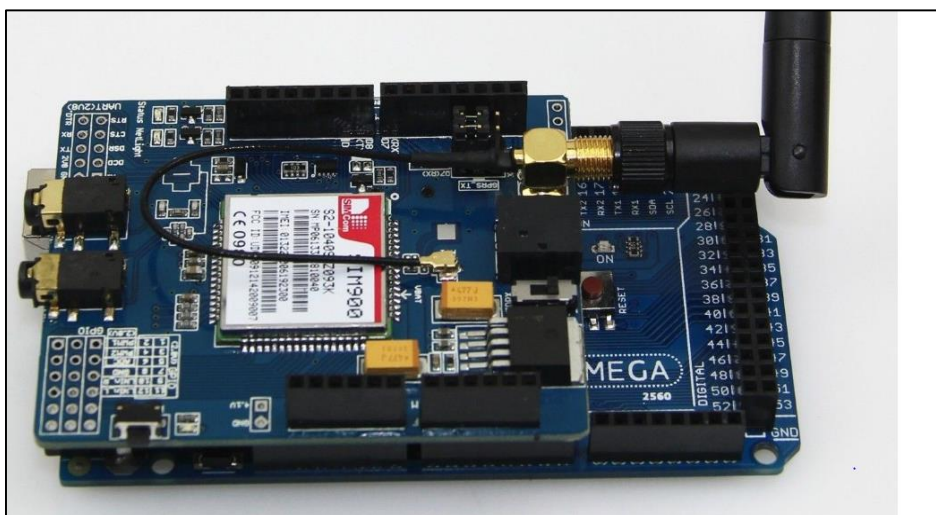
Esquema 7 - Conexões entre o módulo GPS e a placa Arduino



Fonte: o autor.

Quanto ao módulo de comunicação GSM/GPRS SIM900, responsável pelo envio e recebimento de dados entre a sistema embarcado e a estação de monitoramento *web*, o mesmo possui sua pinagem inteiramente compatível com a placa Arduino, e portanto o mesmo se encaixa perfeitamente em cima dele. A fotografia 7 demonstra o seu encaixe.

Fotografia 7 - Encaixe do módulo GSM/GPRS SIM 900 e a placa Arduinno



Fonte: os autores.

O código fonte do *software* embarcado de cada componente da estação meteorológica foi desenvolvido separadamente afim de obter uma melhor organização do código fonte e facilitar a manutenção posterior ao mesmo. Para cada componente de *hardware*, foi criada uma classe separadamente do projeto principal, sendo assim, cada sensor possui um arquivo fonte contendo todas as ações, métodos e atributos utilizados para a coleta das informações. Posteriormente, estas classes foram integradas ao arquivo fonte principal do projeto e assim foi possível montar o fluxo principal do sistema de forma mais clara e objetiva facilitando assim a integração dos componentes de *software* e *hardware*.

Uma boa estratégia utilizada no desenvolvimento do fluxo principal do *software* embarcado foi a utilização do *Design Pattern* Máquina de Estados Finita (*Finite State Machine*). A Máquina de Estados é um padrão de projetos muito conhecido e utilizado no desenvolvimento de *software* para sistemas embarcados, isso porque muitos dispositivos usados em sistemas embarcados são dispositivos passivos, ou seja, respondem a determinadas ações do ambiente. A saída gerada por estas ações depende de seu estado atual, e então aqui entra este *design pattern*: qualquer dispositivo, físico ou conceitual, que possua um conjunto finito de estados e aceita um finito número de entradas, pode produzir um finito número de diferentes saídas.

A implementação da máquina de estados utilizada neste projeto segue o seguinte fluxo:

- 1 Ao “ligar” a estação meteorológica deve inicializar o módulo de comunicação entre a mesma e a estação de monitoramento, para tanto, o estado é alterado para INICIALIZAR\_MODULO;
- 2 Após concluído o módulo de comunicação ter sido inicializado e o mesmo estiver conectado com a estação de monitoramento, a estação está apta a receber as requisições quais solicitam dados atualizados do clima. Para tanto, seu estado é alterado para RECEBER\_COMANDO;
- 3 Quando o estado do sistema embarcado encontra-se no estado de recebimento de comandos, ou seja, está apto ao recebimento de requisições, os pacotes de requisições devem ser analisados identificando qual é a solicitação da estação de monitoramento.
  - a. Caso o pacote de requisição de dados contiver os caracteres “UMIDA”, entende-se que a solicitação é por informações referente à

umidade relativa do ar, e para tanto, o estado do sistema embarcado é alterado para LER\_UMIDADE;

- b. Caso o pacote de requisição de dados contiver os caracteres “GPSLO”, entende-se que a solicitação é por informações referente às coordenadas de localização geográfica, e para tanto, o estado do sistema embarcado é alterado para LER\_LONGITUDE;
- c. Caso o pacote de requisição de dados contiver os caracteres “GPSLA”, entende-se que a solicitação é por informações referente às coordenadas de localização geográfica, e para tanto, o estado do sistema embarcado é alterado para LER\_LATITUDE;
- d. Caso o pacote de requisição de dados contiver os caracteres “LUMIN”, entende-se que a solicitação é por informações referente à luminosidade, e para tanto, o estado do sistema embarcado é alterado para LER\_LUMINOSIDADE;
- e. Caso o pacote de requisição de dados contiver os caracteres “TEMPE”, entende-se que a solicitação é por informações referente à temperatura, e para tanto, o estado do sistema embarcado é alterado para LER\_TEMPERATURA;
- f. Caso o pacote de requisição de dados contiver os caracteres “HISTO”, entende-se que a solicitação é por informações referente às históricos de informações que estão armazenadas da EEPROM, e para tanto, o estado do sistema embarcado é alterado para LER\_HISTORICO;

- 4 Após a interpretação e identificação da requisição recebida, o sistema embarcado estará em um estado específico, conforme detalhado nos subitens ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ e ‘f’ do item 3. Então, o sistema embarcado realizará as devidas tarefas, quais são determinadas pelo seu estado. Quando finalizado a tarefa, o estado do sistema embarcado é alterado para RECEBER\_COMANDO, e então, a nova requisição recebida é avaliada novamente e alterado o estado do sistema embarcado para a qual melhor se identifica.

Para cada estado, foi definido um número inteiro para ela utilizando a diretiva de pré-processador *#define*, esta transforma cada ocorrência da diretiva no código fonte para o número que foi definido a ela. Sendo assim, foi aplicado estrutura *Switch Case* no *loop* principal do projeto tratando a ação de cada estado.



A fotografia 7 demonstra o fluxo principal do código fonte do *software* embarcado.

Fotografia 8 - Fluxo principal do Software Embarcado.

(Continua)

```

1  #include <ldr.h>
2  #include <lm35.h>
3  #include <dht11.h>
4  #include <gprs.h>
5  #include <gps.h>
6
7  GPRS gprs(9600);
8  GPS gps(9600);
9  dht11 DHT11(52);
10 lm35 LM35(A15);
11 ldr LDR(A12);
12
13 String bufferSerial1;
14
15 #define ANENOMETROPIN A11
16
17 //Maquina de estados
18 #define INICIALIZAR_MODULO 0
19 #define RECEBER_COMANDO 1
20 #define LER_UMIDADE 2
21 #define LER_LONGITUDE 3
22 #define LER_LATITUDE 4
23 #define LER_LUMINOSIDADE 5
24 #define LER_TEMPERATURA 6
25 #define LER_HISTORICO 8
26 int estado;
27
28 char* montaPacote(char identificador[], double valor);
29
30 void setup(){
31     Serial.begin(9600);
32     delay(100);
33     estado = INICIALIZAR_MODULO;
34 }
35
36 void loop(){
37     delay(2000);
38     switch(estado){
39     case INICIALIZAR_MODULO:
40         if(gprs.inicializar()){
41             estado = RECEBER_COMANDO;
42         }
43         break;
44         delay(500);
45     case RECEBER_COMANDO:
46         bufferSerial1 = "";
47         bufferSerial1 = gprs.receberComando();
48         Serial.print("Valor lido: ");
49         Serial.println(bufferSerial1);
50         if(bufferSerial1.indexOf("UMIDA") >= 0)
51             estado = LER_UMIDADE;
52         if(bufferSerial1.indexOf("GPSLO") >= 0)
53             estado = LER_LONGITUDE;
54         if(bufferSerial1.indexOf("GPSLA") >= 0)
55             estado = LER_LATITUDE;
56         if(bufferSerial1.indexOf("LUMIN") >= 0)
57             estado = LER_LUMINOSIDADE;
58         if(bufferSerial1.indexOf("TEMPE") >= 0)
59             estado = LER_TEMPERATURA;

```

```

60     if(bufferSerial1.indexOf("HISTO") >= 0)
61         estado = LER_HISTORICO;
62     break;
63 case LER_UMIDADE:
64     Serial.print("Lendo Umidade: ");
65     gprs.enviarComando(montaPacote(":#UM", DHT11.read()));
66     estado = RECEBER_COMANDO;
67     break;
68 case LER_LONGITUDE:
69     Serial.print("Lendo Longitude");
70     gprs.enviarComando(montaPacote(":#LO", gps.getLongitude()));
71     estado = RECEBER_COMANDO;
72     break;
73 case LER_LATITUDE:
74     Serial.print("Lendo Latitude");
75     gprs.enviarComando(montaPacote(":#LA", gps.getLatitude()));
76     estado = RECEBER_COMANDO;
77     break;
78 case LER_LUMINOSIDADE:
79     Serial.print("Lendo Luminosidade");
80     gprs.enviarComando(montaPacote(":#LU", LDR.read()));
81     estado = RECEBER_COMANDO;
82     break;
83 case LER_TEMPERATURA:
84     Serial.println("Lendo Temperatura");
85     gprs.enviarComando(montaPacote(":#TP", LM35.read()));
86     estado = RECEBER_COMANDO;
87     break;
88 case LER_HISTORICO:
89     Serial.println("Lendo Historico");
90     estado = RECEBER_COMANDO;
91     break;
92 }
93 }
94
95 char* montaPacote(char identificador[], double valor){
96     char convert[15];
97     sprintf(convert, "%.8f", valor);
98     char pacote[20];
99     strcat(pacote, identificador);
100    strcat(pacote, convert);
101    Serial.println(pacote);
102    return pacote;
103 }

```

Fonte: os autores.

O módulo GPS disponibiliza um pacote de informações bastante grande, onde contém além das coordenadas geográficas, diversas outras informações como. Para este projeto, nos interessa apenas a sentença *GPGGA* (*Global Positioning System Fix Data*) qual contém as informações referentes às coordenadas geográficas.

O Google *Maps* utiliza um formato diferente do padrão de coordenadas enviado pelo GPS, para tanto, é necessária realizar uma conversão dos dados fornecidos do GPS a fim de obter as coordenadas de latitude e longitude. Esta conversão é bastante simples, separa-se uma parte dos valores de latitude e longitude, e após isso soma-as e divide-se por 60. Após isso, verifica-se o hemisfério, caso o mesmo contenha a letra “S” ou “W”

deve-se multiplicar o resultado da conta anterior por “-1”. A tabela 2 demonstra esta conversão com maiores detalhes.

Tabela 2 – Conversão para Obtenção das Coordenadas válidas ao Google *Maps*.

Sentença NMEA				
\$GPGGA,014654.000,2709.9870,S,05131.0290,W,1,9,1.34,643.9,M,3.1,M,,*63.				
	Latitude	Hemisfério	Longitude	Hemisfério
Dados fornecidos pelo GPS	2709.9870	S	05131.0290	W
Conversão matemática	27 + (09.9870)	-1	51 + (31.0290)	-1

O resultado obtido desta conversão de exemplo foi:

Latitude	Longitude
-27.16645	-51.51715

Fonte: os autores.

A fotografia 9 apresenta o código fonte implementado para realizar a seleção somente da sentença GPGGA e demais rotinas para realizar a conversão dos dados em com coordenadas válidas para o Google *Maps*.

Fotografia 9 – Implementação da classe responsável pela coleta da Localização Geográfica. (Continua)

```

1  #include "gps.h"
2  #include "Arduino.h"
3
4  GPS::GPS(int baudrate){
5      Serial2.begin(baudrate);
6  }
7  double GPS::getLatitude(){
8      obterDados();
9      return latitude;
10 }
11 double GPS::getLongitude(){
12     obterDados();
13     return longitude;
14 }
15 void GPS::obterDados(){
16     char entrada = 0;
17     String resposta = "";
18     while((entrada = Serial2.read()) != 10) {
19         if(entrada > 0)
20             resposta += entrada;
21     }
22     if(!resposta.equals("")){
23         String teste = resposta.substring(3, 6);
24         if(teste == "GGA"){
25             //LATITUDE
26             char *latitudeStr = subString(resposta, 19, 27);
27             latitude = atof(latitudeStr);
28             char *ajusteLat = subString(resposta, 17, 19);
29             int ajusteLatInt = atoi(ajusteLat);
30             latitude = ((latitude/60) + ajusteLatInt);
31             int cardealLat;
32             if(resposta[28] == 'N')
33                 cardealLat = 1;
34             else
35                 cardealLat = -1;
36             //Ajusta a latitude conforme o hemisfério
37             latitude *= cardealLat;

```

Fotografia 9 – Implementação da classe responsável pela coleta da Localização Geográfica. (Conclusão)

```

38         //LONGITUDE
39         char *longitudeStr = subString(resposta, 33, 41);
40         longitude = atof(longitudeStr);
41         char *ajusteLon = subString(resposta, 30, 33);
42         int ajusteLonInt = atoi(ajusteLon);
43         longitude = ((longitude/60) + ajusteLonInt);
44         int cardealLon;
45         if(resposta[42] == 'E')
46             cardealLon = 1;
47         else
48             cardealLon = -1;
49         //Ajusta a longitude conforme hemisfério
50         longitude *= cardealLon;
51     }
52 }
53 }
54
55 char* GPS::subString(String mensagem, int posInicial, int posFinal){
56     int tamanho = posFinal - posInicial + 1;
57     char temp[tamanho];
58     int posicao = 0;
59     for(int i = posInicial; i < posFinal; i++){
60         temp[posicao] = mensagem[i];
61         posicao++;
62     }
63     temp[posicao] = '\0';
64     return temp;
65 }
66

```

Fonte: os autores.

A implementação desenvolvida para a realização da criação da conexão entre o sistema embarcado e a estação de monitoramento, é a demonstrada na Fotografia 11. Esta, contém já programados os comandos AT que são responsáveis pela criação de uma Conexão Única (*Single Connection*) de modo Transparente através do protocolo TCP/IP.

Fotografia 10 – Implementação da classe responsável pela criação da conexão TCP/IP. (Continua)

```

1  #include "gps.h"
2
3  GPRS::GPRS(int baudrate){
4      Serial1.begin(baudrate);
5  }
6
7  String GPRS::receberComando(){
8      String bufferSerial = String(128);
9      Serial1.flush();
10     bufferSerial = "";
11     delay(2000);
12     while(Serial1.available() > 0){
13         bytes = Serial1.read();
14         bufferSerial = bufferSerial + bytes;
15         delay(200);
16     }
17     Serial.println(bufferSerial);
18     return bufferSerial;
19 }
20
21 void GPRS::enviarComando(char *comando){
22     Serial1.println(comando);
23 }
24

```

Fotografia 10 – Implementação da classe responsável pela criação da Conexão TCP/IP. (Conclusão)

```

21 void GPRS::enviarComando(char *comando){
22     Serial1.println(comando);
23 }
24
25 bool GPRS::inicializar(){
26     /*Inicialização da configuração do módulo GPRS*/
27     String resposta = "";
28     do{
29         enviarComando("AT+CIPMUX=0");
30         delay(1000);
31         resposta = "";
32         resposta = receberComando();
33     }while(resposta.indexOf("OK") < 2);
34     do{
35         enviarComando("AT+CIPMODE=1");
36         delay(1000);
37         resposta = receberComando();
38     }while(resposta.indexOf("OK") < 2);
39     do{
40         enviarComando("AT+CGATT?");
41         delay(2000);
42         resposta = receberComando();
43     }while(resposta.indexOf("OK") < 2);
44     do{
45         enviarComando("AT+CSTT=\"zap.vivo.com.br\", \"vivo\", \"vivo\"");
46         delay(1000);
47         resposta = receberComando();
48     }while(resposta.indexOf("OK") < 2);
49     do{
50         enviarComando("AT+CIICR");
51         delay(6000);
52         resposta = receberComando();
53     }while(resposta.indexOf("OK") < 2);
54     do{
55         enviarComando("AT+CIFSR");
56         delay(5000);
57         resposta = receberComando();
58     }while(resposta.indexOf(".") < 1);
59     do{
60         enviarComando("AT+CIPSTART=\"tcp\", \"200.193.93.7\", \"6000\"");
61         delay(5000);
62         resposta = receberComando();
63     }while(resposta.indexOf("CONNECT") < 7);
64     return true;
65 }

```

Fonte: os autores.

As Fotografias 11, 12 e 13 refere-se ao código fonte das dos sensores responsáveis pela coleta de dados da estação meteorológica: temperatura, luminosidade e umidade relativa do ar, respectivamente.

Fotografia 11 – Implementação da classe responsável pela coleta da Temperatura.

```

1 #include "lm35.h"
2
3 lm35::lm35(int pin){
4     this->pin = pin;
5 }
6
7 double lm35::read(){
8     pinMode(pin, INPUT);
9     temperatura = (analogRead(pin) * 0.00488)*100;
10    return temperatura;
11 }
12

```

Fonte: os autores.

Fotografia 12 - Implementação da classe responsável pela coleta da Luminosidade.

```

1  #include "ldr.h"
2
3  LDR::LDR(int pin){
4      this->pin = pin;
5  }
6
7  double LDR::read()
8  {
9      int cont, i;
10     pinMode(pin, INPUT);
11     luminosidade = 0;
12     for(i=0; i<=10; i++){
13         cont=analogRead(pin);
14         luminosidade = luminosidade + cont;
15         delay(10);
16     }
17     luminosidade /= 10;
18     return luminosidade;
19 }

```

Fonte: os autores.

Fotografia 13 – Implementação da classe responsável pela coleta da Umidade.

```

1  #include "dht11.h"
2
3  DHT11::DHT11(int pin){
4      this->pin = pin;
5  }
6
7  double DHT11::read()
8  {
9      uint8_t bits[5];
10     uint8_t cnt = 7;
11     uint8_t idx = 0;
12
13     for (int i=0; i< 5; i++) bits[i] = 0;
14
15     pinMode(pin, OUTPUT);
16     digitalWrite(pin, LOW);
17     delay(18);
18     digitalWrite(pin, HIGH);
19     delayMicroseconds(40);
20     pinMode(pin, INPUT);
21
22     unsigned int loopCnt = 10000;
23     while(digitalRead(pin) == LOW)
24         if (loopCnt-- == 0) return -2;
25
26     loopCnt = 10000;
27     while(digitalRead(pin) == HIGH)
28         if (loopCnt-- == 0) return -2;
29
30     for (int i=0; i<40; i++)
31     {
32         loopCnt = 10000;
33         while(digitalRead(pin) == LOW)
34             if (loopCnt-- == 0) return -2;
35
36         unsigned long t = micros();
37
38         loopCnt = 10000;
39         while(digitalRead(pin) == HIGH)
40             if (loopCnt-- == 0) return -2;
41
42         if ((micros() - t) > 40) bits[idx] |= (1 << cnt);
43         if (cnt == 0)
44         {
45             cnt = 7;
46             idx++;
47         }
48         else cnt--;
49     }
50     humidity = bits[0];
51     return humidity;
52 }

```

Fonte: os autores.

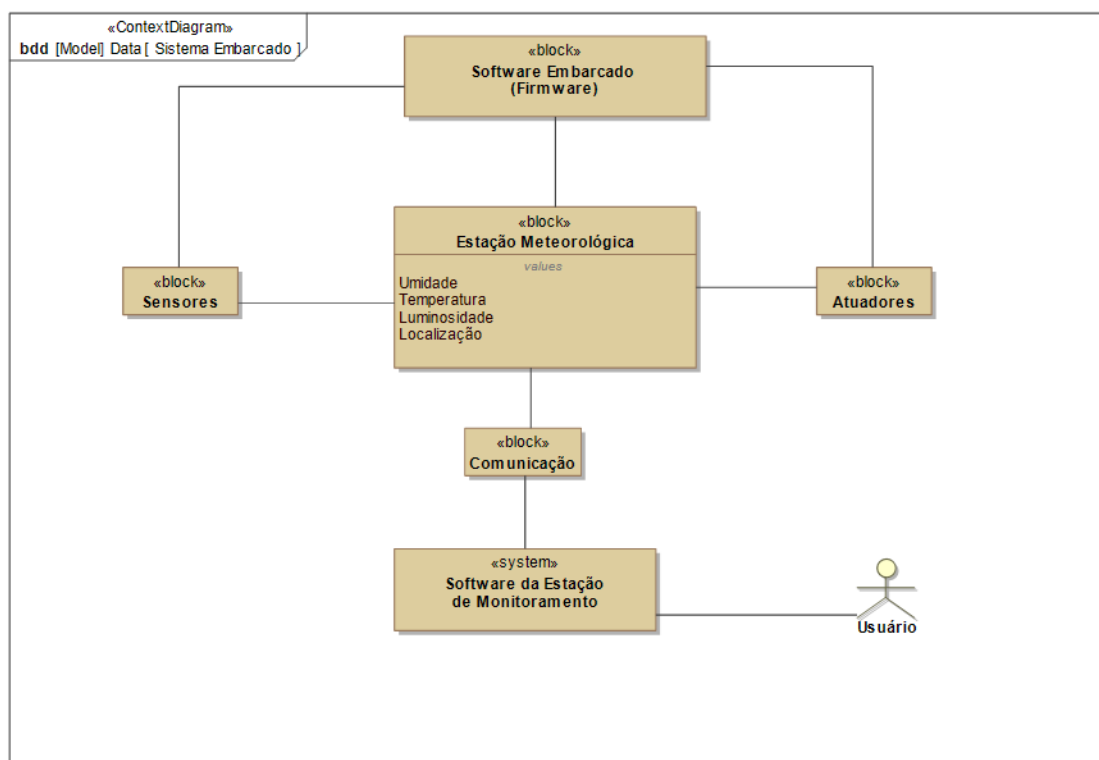
### 3.4 DESENVOLVIMENTO DO SISTEMA SUPERVISÓRIO

Nas próximas sessões são destinadas a demonstrar como foi realizada a construção do sistema de monitoramento *web*, o qual é responsável por demonstrar graficamente dados do tempo coletadas pela estação meteorológica e também requisitar dados atualizados de tempos em tempos a estação.

#### 3.4.1 Engenharia de Requisitos e Especificações

Para esta etapa, continuou-se utilizando os diagramas da SysML como forma de documentação e para auxiliar na organização e desenvolvimento do sistema web. O Diagrama 5 demonstra o escopo do projeto. Este foi projetado utilizando o diagrama de contexto da SysML.

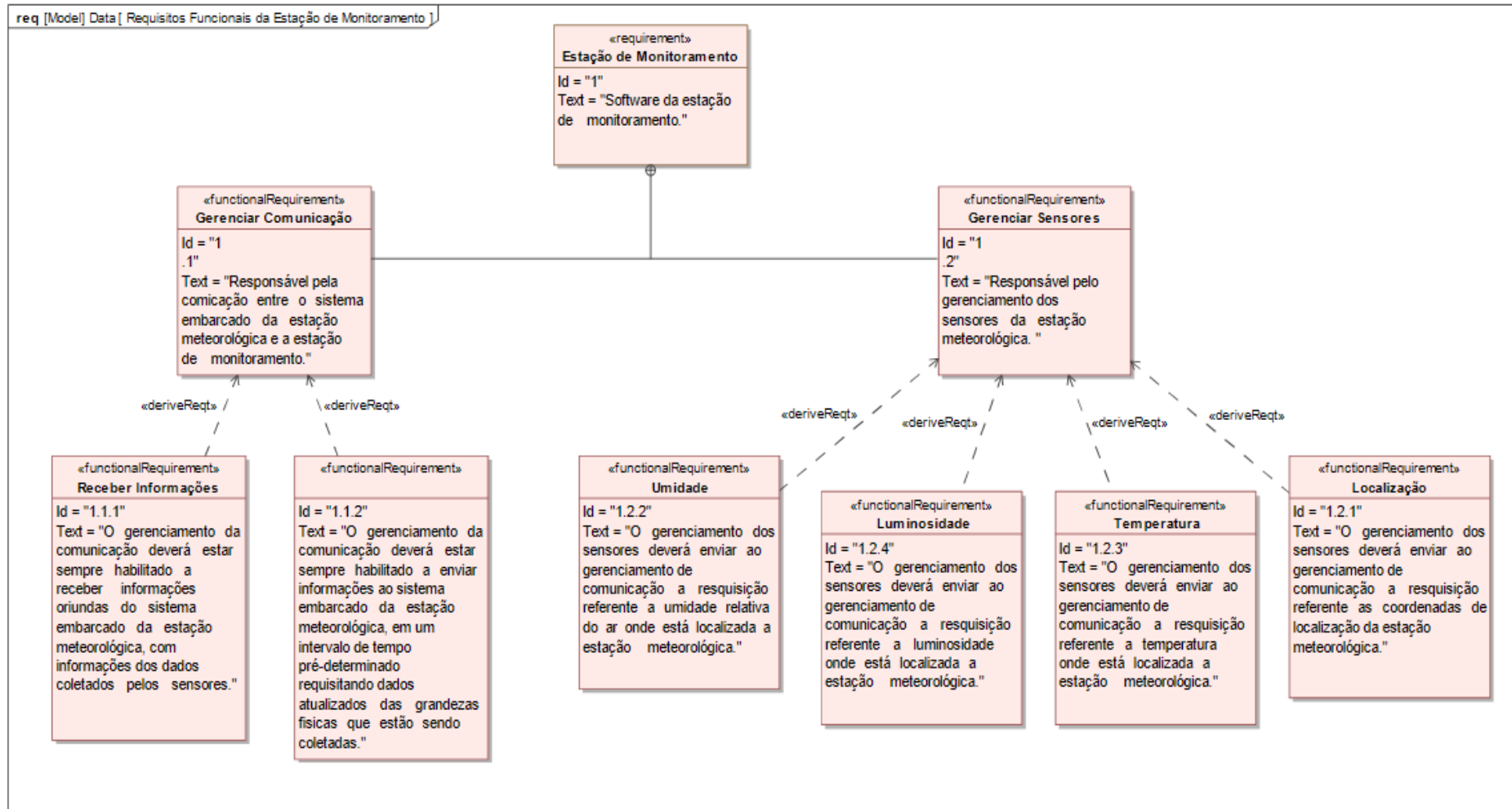
Diagrama 6 - Escopo do Projeto.



Fonte: os autores.

O levantamento dos requisitos funcionais do software é demonstrado através do diagrama de requisitos da SysML, como também foi realizado na modelagem do projeto do sistema embarcado.

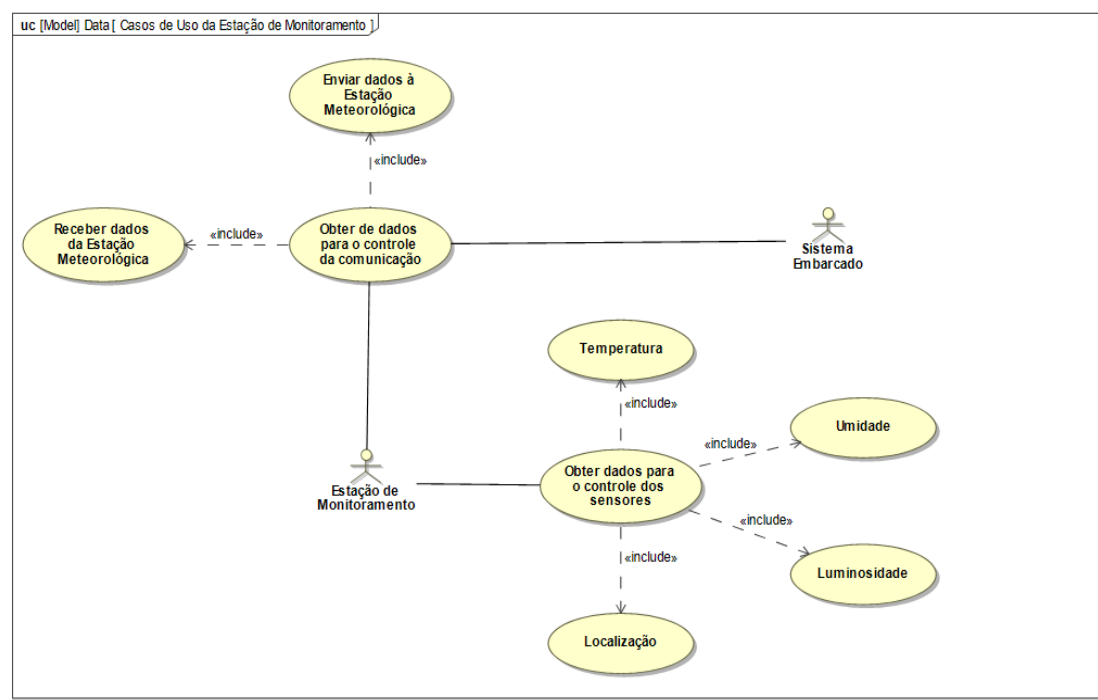
Diagrama 7 - Levantamento dos Requisitos Funcionais



Fonte: os autores.



Diagrama 8 - Diagrama de Casos de Uso da Estação de Monitoramento.



Fonte: os autores.

### 3.4.2 Integração dos Dispositivos e Sistemas

Nesta etapa é realizado o desenvolvimento do sistema de monitoramento *web*. Como requisito parcial deste trabalho, é necessário que o mesmo realize o envio de requisições a estação meteorológica, solicitando dados atualizados através de *sockets*. O *socket* é uma forma de comunicação na qual permite a troca de dados entre aplicações que utilizam apenas *software* e entre dispositivos de *hardware* e *software* utilizando uma estrutura de rede de computadores. *Sockets* são compostos pela combinação de um endereço de IP e porta.

Para que isso fosse efetuado de forma automática, foi utilizado o recurso de *Timer* e *TimerTask*, da biblioteca “*java.util.Timer*” e “*java.util.TimerTask*”, estas duas bibliotecas permitem que uma tarefa ou ação seja executada durante um intervalo de tempo, que é determinado pelo programador na própria função. Junto a isso, foi criada uma *Thread* para ficar executando a *TimerTask* evitando que a aplicação fique travada durante o envio das requisições.

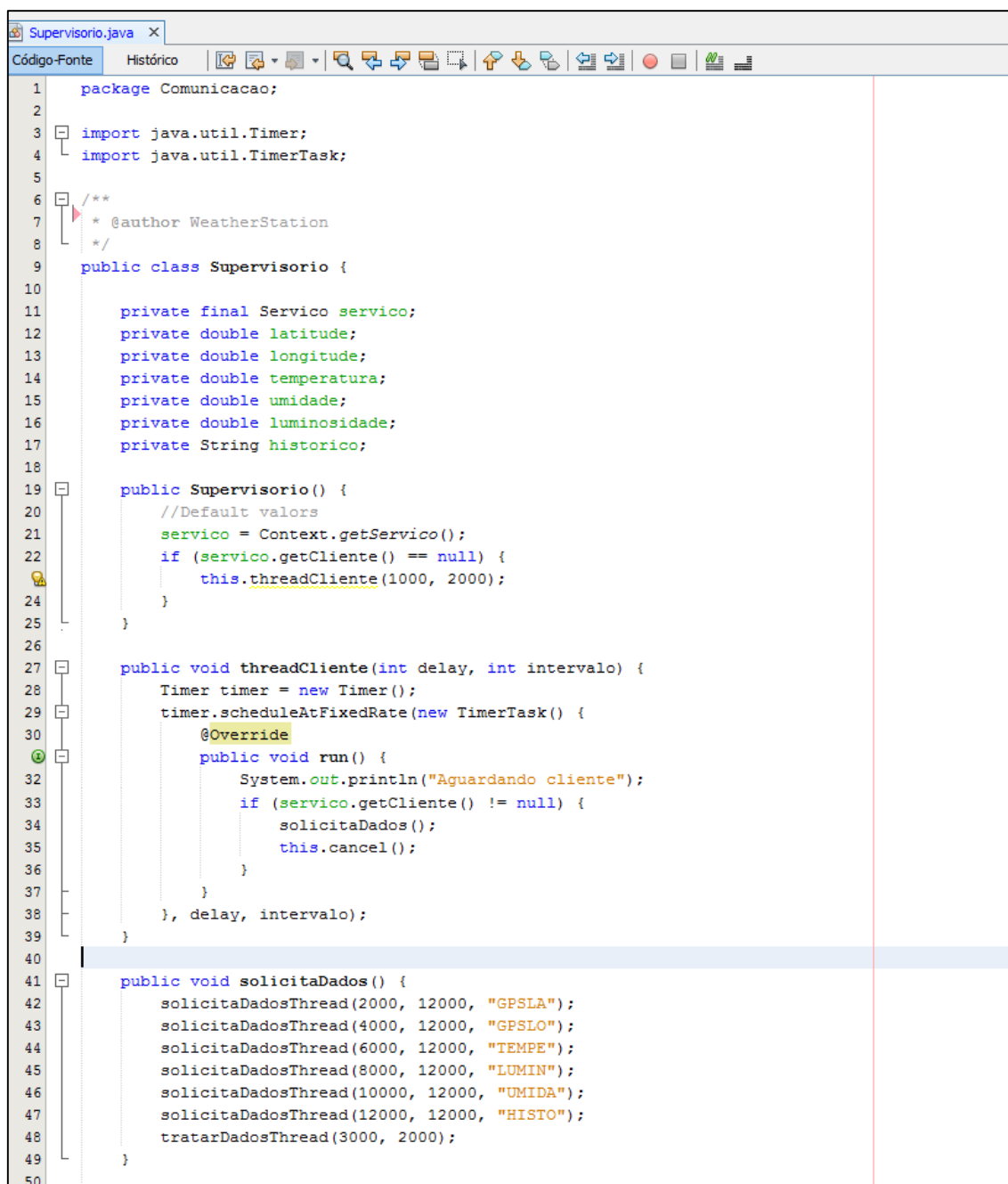
Quando é recebido as respostas das requisições efetuadas, é invocado a *thread* responsável por tratar os dados, os quais irão atribuir os valores recebidos aos atributos das classes de regra de negócio do projeto. Sendo assim, a página *web* criada apenas irá

buscar os atributos das classes de regra de negócio, através dos modificadores de acesso destas e exibindo as informações que estão armazenadas nos atributos.

Na Fotografia 14 é possível visualizar a classe Supervisorio, onde a mesma implementa os métodos: *threadCliente()* que após estabelecida a conexão entre o sistema supervisorio e a estação meteorológica invoca o método *solicitaDados()* responsável por realizar o envio das requisições à estação meteorológica. O método *tratarDados()* é chamado no intervalo de tempo do envio das requisições, e é responsável por interpretar o pacote de dados recebido da estação e extrair as informações relevantes e armazena-las.

Fotografia 14 – Implementação da classe Supervisorio.

(Continua)



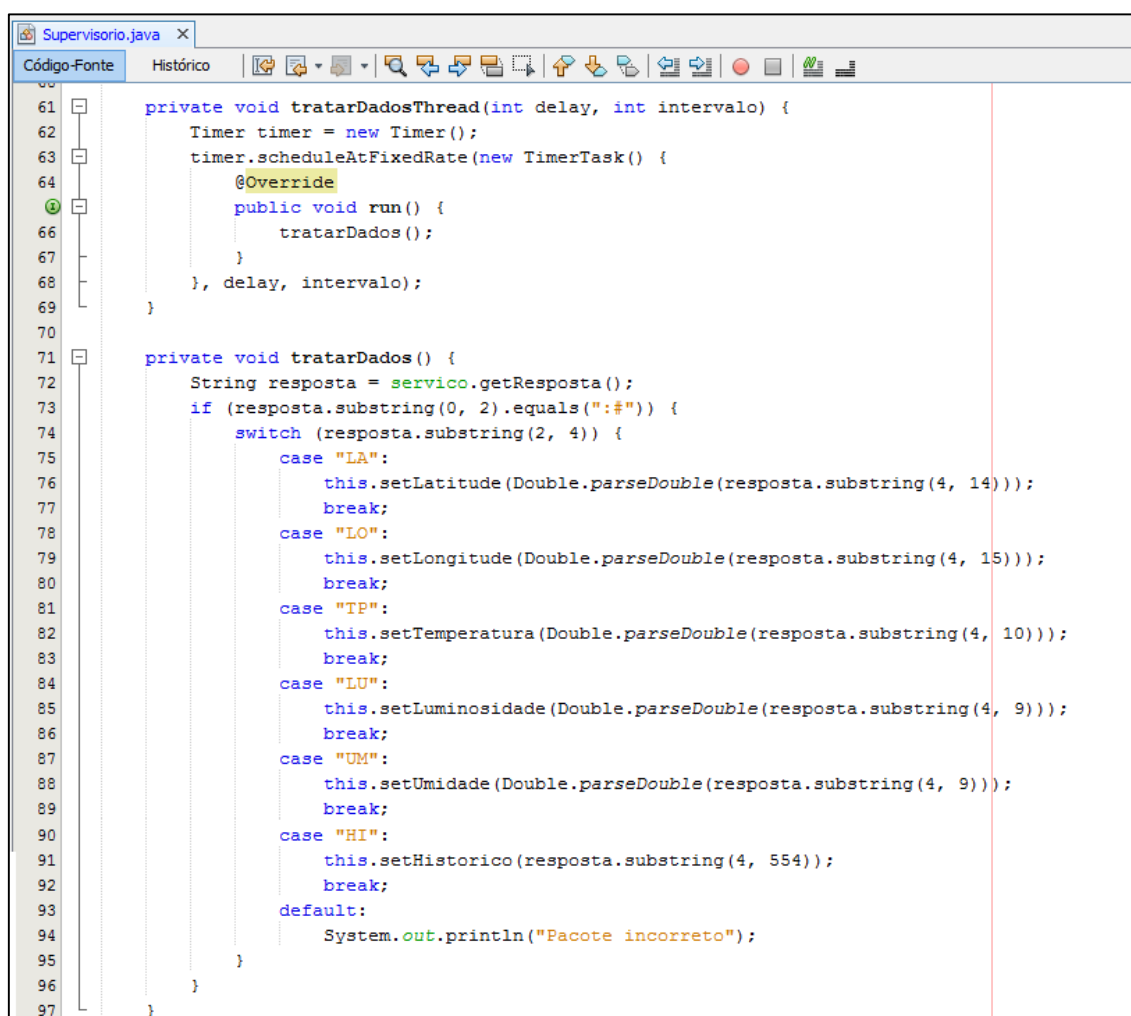
```

1 package Comunicacao;
2
3 import java.util.Timer;
4 import java.util.TimerTask;
5
6 /**
7  * @author WeatherStation
8  */
9 public class Supervisorio {
10
11     private final Servico servico;
12     private double latitude;
13     private double longitude;
14     private double temperatura;
15     private double umidade;
16     private double luminosidade;
17     private String historico;
18
19     public Supervisorio() {
20         //Default valores
21         servico = Context.getServico();
22         if (servico.getCliente() == null) {
23             this.threadCliente(1000, 2000);
24         }
25     }
26
27     public void threadCliente(int delay, int intervalo) {
28         Timer timer = new Timer();
29         timer.scheduleAtFixedRate(new TimerTask() {
30             @Override
31             public void run() {
32                 System.out.println("Aguardando cliente");
33                 if (servico.getCliente() != null) {
34                     solicitaDados();
35                     this.cancel();
36                 }
37             }
38         }, delay, intervalo);
39     }
40
41     public void solicitaDados() {
42         solicitaDadosThread(2000, 12000, "GPSLA");
43         solicitaDadosThread(4000, 12000, "GPSLO");
44         solicitaDadosThread(6000, 12000, "TEMPE");
45         solicitaDadosThread(8000, 12000, "LUMIN");
46         solicitaDadosThread(10000, 12000, "UMIDA");
47         solicitaDadosThread(12000, 12000, "HISTO");
48         tratarDadosThread(3000, 2000);
49     }
50

```

Fotografia 14 – Implementação da classe Supervisorio.

(Conclusão)



```

61 private void tratarDadosThread(int delay, int intervalo) {
62     Timer timer = new Timer();
63     timer.scheduleAtFixedRate(new TimerTask() {
64         @Override
65         public void run() {
66             tratarDados();
67         }
68     }, delay, intervalo);
69 }
70
71 private void tratarDados() {
72     String resposta = servico.getResposta();
73     if (resposta.substring(0, 2).equals(":#")) {
74         switch (resposta.substring(2, 4)) {
75             case "LA":
76                 this.setLatitude(Double.parseDouble(resposta.substring(4, 14)));
77                 break;
78             case "LO":
79                 this.setLongitude(Double.parseDouble(resposta.substring(4, 15)));
80                 break;
81             case "Tp":
82                 this.setTemperatura(Double.parseDouble(resposta.substring(4, 10)));
83                 break;
84             case "LU":
85                 this.setLuminosidade(Double.parseDouble(resposta.substring(4, 9)));
86                 break;
87             case "UM":
88                 this.setUmidade(Double.parseDouble(resposta.substring(4, 9)));
89                 break;
90             case "HI":
91                 this.setHistorico(resposta.substring(4, 554));
92                 break;
93             default:
94                 System.out.println("Pacote incorreto");
95         }
96     }
97 }

```

Fonte: os autores.

Em um intervalo de tempo pré-definido, todos os componentes gráficos da página *web* são atualizados automaticamente sendo capazes de exibirem os novos valores coletados pela estação meteorológica. A interface foi construída utilizando o *framework* JSF e recursos do PrimeFaces 5.0, que dão um visual mais elegante a aplicação e deixando-a mais interativa, com efeitos e temas.

## 5 TRABALHOS RELACIONADOS

Devido ao prazo de entrega deste trabalho ter sido postergada, foi realizada a implementação de um anemômetro, qual é responsável por medir a velocidade do vento. Esta implementação não estava prevista ao projeto, portanto, a mesma não foi devidamente planejada seguindo a metodologia que a engenharia de software indica. Para mais detalhes deste trabalho, pode ser verificado a seção de anexos, item “A” onde é

demonstrado como foi confeccionado artesanalmente o anemômetro e como foi integrado ao sistema embarcado.

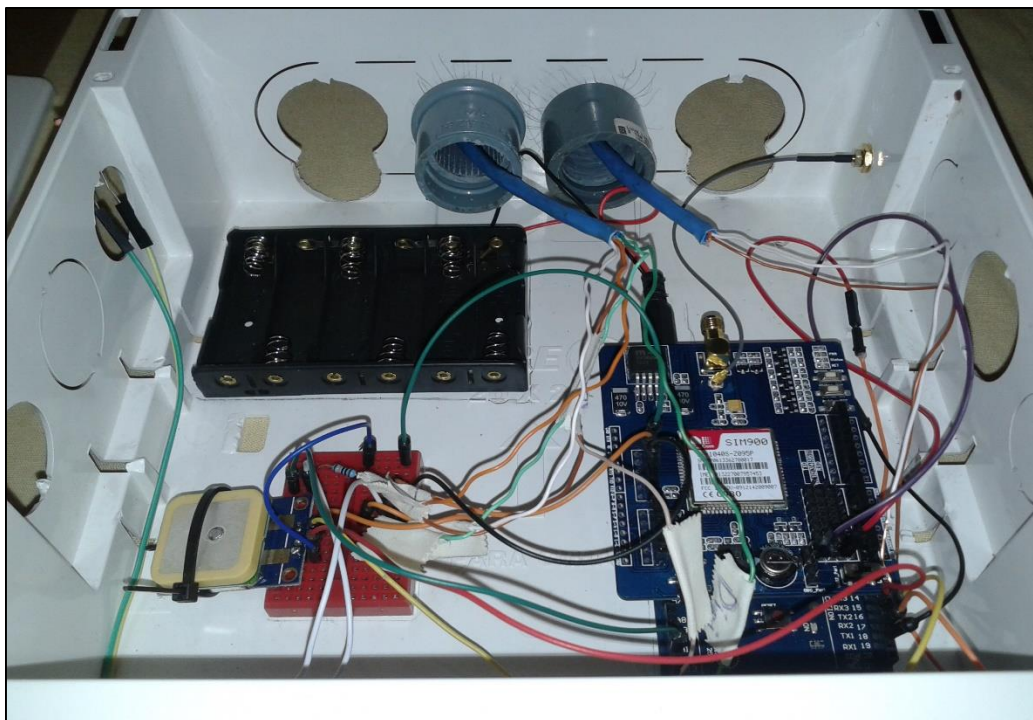
## 6 RESULTADOS OBTIDOS

Nesta seção são apresentados os resultados do projeto realizado, demonstrando o sistema embarcado e o sistema *web* que foram desenvolvidos no decorrer do projeto.

Na fotografia 15 pode-se visualizar a primeira etapa do projeto, que é o desenvolvimento do sistema embarcado da estação meteorológica. Como um adicional ao projeto, pode ser visualizado que o mesmo encontra-se em uma caixa de PVC, para proteger os componentes de *hardware* que estão dentro da mesma. Apenas a antena do módulo GSM/GPRS é colocada para fora da caixa.

A caixa ainda possui dois canos que atravessam a mesma, na extremidade superior de um destes canos foi colocado o sensor de umidade DHT11 e seus cabos descem pelo cano e são conectados a placa Arduino. Já o outro cano foi utilizado para a fixação de um Anemômetro, equipamento capaz de medir a velocidade do vento. Este item foi desenvolvido a parte do projeto, e por isso não constam detalhes de sua implementação nas seções anteriores. Mais detalhes deste item podem ser visualizados no Anexo A.

Fotografia 15 - Sistema Embarcado montado na Estação Meteorológica

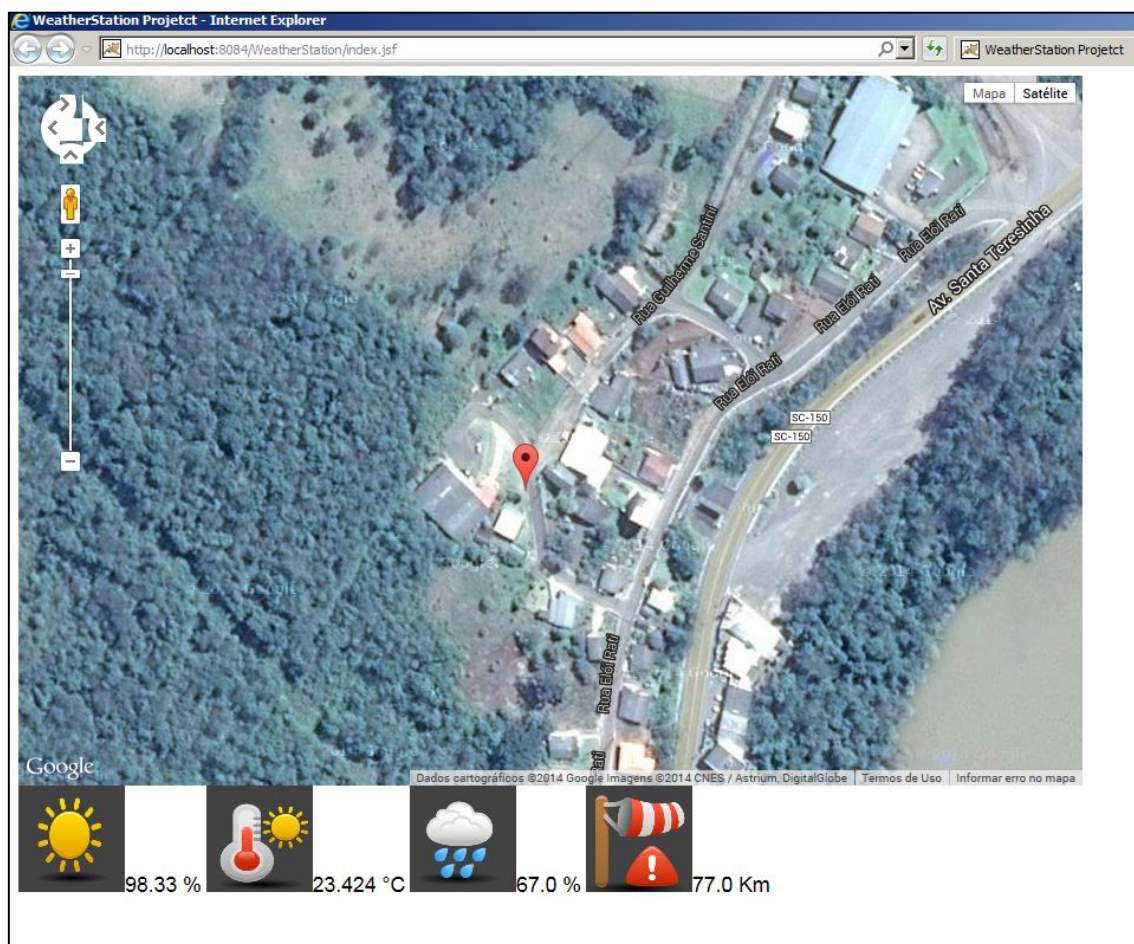


Fonte: os autores.



A segunda etapa do projeto é composta pelo desenvolvimento da estação de monitoramento, que é uma aplicação *web*. A Fotografia 16 demonstra a página desenvolvida, onde é possível visualizar os componentes gráficos utilizados para demonstrar os dados coletados pela estação meteorológica de uma forma bastante dinâmica.

Fotografia 16 - Interface do Sistema Web



Fonte: os autores.

Dependendo do valor dos dados coletados da estação meteorológica, os ícones que indicam Temperatura, Luminosidade, Umidade e Vento são alterados. Isso somente é permitido graças a atualização que ocorre no componente gráfico de tempos em tempos. O mapa também exibe a localização atual da estação meteorológica, o local é indicado por um *Marker*, que também é disponibilizado pelos componentes da API do Prime Faces.

É possível visualizar também informações de dados coletados das últimas 24h pelo gráfico de linha que é apresentado pela aplicação. Os valores são armazenados na EEPROM da placa Arduino em um intervalo de tempo de 1 hora cada. Quando solicitado

pela aplicação, é enviado um pacote de dados com todos os dados armazenados e então estes dados são demonstrados no gráfico. Cada série do mesmo equivale a uma grandeza física, que pode ser identificada pela legenda.

Fotografia 17 - Protótipo da Estação Meteorológica montada



Fonte: os autores.

## 7 DISCUSSÕES

A respeito do sistema embarcado desenvolvido, algumas considerações importantes são necessárias em relação a placa de prototipação Arduino. Para o desenvolvimento do protótipo da estação meteorológica proposta neste projeto, a mesma demonstrou um bom desempenho e estabilidade. Porém, esta placa é indicada apenas para prototipação de projetos. A aplicação deste projeto em situações nas quais precisa-se de uma maior precisão e processamento de dados, se faz necessário repensar na utilização da mesma.

Quanto a forma de comunicação utilizada, deve é necessário ressaltar que a estação meteorológica deve estar localizada em uma área onde há sinal de celular, também, há a necessidade de adesão de um plano de dados móveis, capaz de suportar o tráfego de dados diário/mensal, pois, caso o limite de dados seja excedido, o tráfego das informações não será mais efetuado e portanto, não haverá integridade e confiabilidade dos dados exibidos pela aplicação *web*.

As modelagens desenvolvidas, serviram como base para a implementação do projeto e também propiciaram a todos os integrantes da equipe uma maior clareza nas definições do escopo e limites do projeto. Não foram estudados a fundo as técnicas e padrões no desenvolvimento de sistemas embarcados que são abordados pela Engenharia de Software, pois não é este o foco da disciplina.

Ao que se refere ao anemômetro, este foi desenvolvido como um adicional ao projeto, assim como a estrutura da estação meteorológica. Nenhum destes itens eram requisitos do projeto, portanto neste trabalho não a menção dos mesmos nas modelagens do projeto nem no referencial teórico ao projeto. Detalhes desta implementação forma adicionadas aos anexos deste documento.

## 8 CONCLUSÃO

Os resultados obtidos com a realização deste trabalho foram satisfatórios. Foi possível implementar tanto o sistema embarcado, juntamente com o firmware para seu controle e também o sistema web. O planejamento e as modelagens implementadas auxiliaram na implementação e integração do sistema, pois com isso foi possível prever a necessidade de recursos, problemas que iríamos ter.

Para o desenvolvimento deste projeto, houve toda uma pesquisa por referencial teórico qual foi essencial para o entendimento do funcionamento dos componentes e recursos utilizados. O estudo dos componentes contribuiu para a correta utilização dos mesmos não gerando problemas durante sua implementação.

Contudo, todo o estudo, desenvolvido e a implementação deste trabalho contribuíram para o entendimento do conteúdo da disciplina, e nos deixaram mais preparados para a implementação de trabalhos futuros.



## 9 ANEXOS

### ANEXO A – ANEMÔMETRO

O nome anemômetro vem do grego “Anemós” que significa vento e é um equipamento utilizado para medir a velocidade dos ventos em um sistema meteorológico. Os tipos mais conhecidos de anemômetros são os de conchas e os de hélice, ambos partem pelo mesmo princípio de medição da velocidade do vento, em quilômetros por hora.

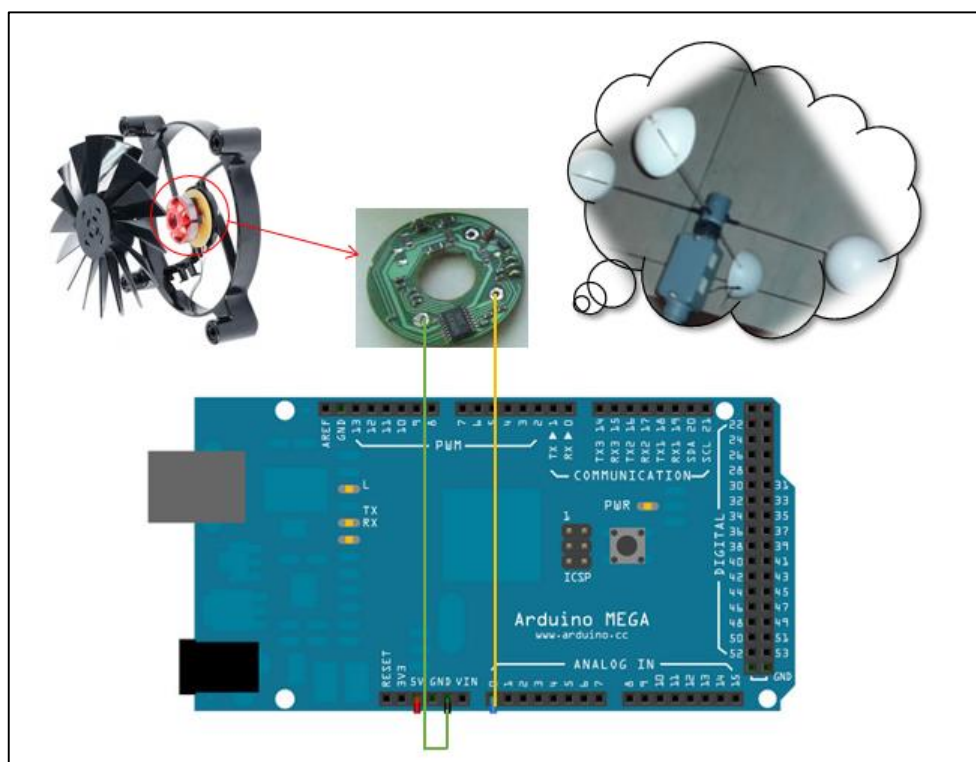
Neste projeto, realizamos a implementação de um anemômetro na estação meteorológica. Este equipamento foi desenvolvido pelos integrantes da equipe, utilizando materiais que tínhamos em casa para a confecção do mesmo. Abaixo segue a relação dos itens utilizados:

- Cola quente;
- Cola instantânea extra forte;
- Duas bolas de plástico (serradas ao meio formando conchas);
- Raios de bicicleta;
- *Cooler* de computador;
- Pedacos de canos;
- Cintas plásticas;

Para que o equipamento pudesse girar na velocidade do vento utilizamos um *cooler* para ser nossa base. No *cooler* usado retiramos as hélices deixando somente o plástico que reveste a bobina. Para fixar as conchas, foi necessário furar o cano em quatro lugares para que os raios se cruzassem dentro do mesmo. Preenchemos o mesmo de cola quente para que o cano não se movimentasse e amarramos nas extremidades com cintas plásticas. Depois de furado o cano e inserido as conchas com os raios, foi colado com a cola instantânea em cima do plástico que reveste a bobina. Para saber qual a velocidade que o protótipo está, desmontamos a bobina junto com o circuito e retiramos os fios de VCC e GND, soldando outros dois fios na saída da bobina, uma em cada lado. Com isso a energia eólica é transformada em energia elétrica através das conchas, sendo que sua tensão elétrica máxima é de 5 *volts*.

Depois de finalizado para calibrar o equipamento, foi colocada a mesma no carro, onde verificamos a velocidade do momento e o valor emitido pelo conversor A/D do Arduino, qual realizava a conversão dos valores analógicos para digital, com isso realizamos uma escala para demonstrar em nosso trabalho.

Fotografia 18 - Esquema para a confecção do anemômetro.



Fonte: os autores.

## REFERÊNCIAS

- APRENDER ELETRÔNICA. **O que é LDR?**. [2012?]. Disponível em: <<http://www.aprenderelectronica.com.br/o-que-e-ldr-resistor-dependente-de-luz>>. Acesso em: 01 jul. 2014.
- ALBUQUERQUE, Paulo César Gurgel. SANTOS, Cláudia Cristina. GPS para iniciantes. Instituto Nacional de Pesquisas Espaciais INPE: Simpósio Brasileiro de Sensoriamento. São José dos Campos, 2003. Disponível em: <<http://geosenso.com/arquivos/GPS%20para%20iniciantes%20-%20INPE.pdf>>. Acesso em: 20 jun. 2014
- ARAUJO, João Gualberto R.: **Boletim bimestral sobre tecnologia de redes**. Rede Nacional de Ensino e Pesquisa (RNP). 1997. Disponível em: <<http://www.rnp.br/newsgen/9710/n5-3.html>> Acesso em: 9 mai. 2014.
- ASSEF, Amauri. **Tópicos Especiais Em Eletrônica Industrial - Sistemas Embarcados**. [2013?]. **Notas de aula**. Disponível em: <[http://paginapessoal.utfpr.edu.br/amauriassef/disciplinas/topicos-especiais-em-eletronica-industrial/Apresentacao\\_Sistemas\\_Embarcados\\_1.pdf/view](http://paginapessoal.utfpr.edu.br/amauriassef/disciplinas/topicos-especiais-em-eletronica-industrial/Apresentacao_Sistemas_Embarcados_1.pdf/view)>. Acesso em: 15 mar. 2014.
- ARDUINO. **Arduino Mega**. [2005?]. Disponível em: <<http://arduino.cc/en/Main/arduinoBoardMega>>. Acesso em: 18 abr. 2014.
- CENTRO ESTADUAL DE METEOROLOGIA – CEMETRS. **Estações Meteorológicas Automáticas**. [2013?]. Disponível em: <[http://www.cemet.rs.gov.br/conteudo/412/?Esta%C3%A7%C3%A3o\\_meteorol%C3%B3gica\\_autom%C3%A1tica](http://www.cemet.rs.gov.br/conteudo/412/?Esta%C3%A7%C3%A3o_meteorol%C3%B3gica_autom%C3%A1tica)>. Acesso em: 01 jun. 2014.
- D-Robotics UK. **DHT11 Humidity & Temperature Sensor**. [2014?]. Disponível em: <[www.droboticsonline.com](http://www.droboticsonline.com)>. Acesso em: 01 jul. 2014.
- ELECFREAKS. **Frearduino Mega2560 - Specification**. 2012. Disponível em: <[http://www.electfreaks.com/wiki/index.php?title=Frearduino\\_Mega2560&oldid=2962](http://www.electfreaks.com/wiki/index.php?title=Frearduino_Mega2560&oldid=2962)>. Acesso em: 19 abr. 2014.
- FRANÇA, Flávio Almada. **Tutorial – Entendendo Java para Web (Parte 1)**. 2010. Disponível em: <<http://flavioaf.wordpress.com/2010/02/25/tutorial-entendendo-java-para-web-parte-1/>>. Acesso em: 11 mai. 2014.
- INSTITUTO NACIONAL DE METEOROLOGIA - INMET. **Estações Convencionais**. [2013?]. Disponível em: <<http://www.inmet.gov.br/portal/index.php?r=estacoes/estacoesConvencionais>>. Acesso em: 01 jun. 2014.
- MACHADO, Marcus Vinicius Ribeiro. **Sistema embarcado sem fio para monitoramento de sinais em soldagem a arco elétrico com abordagem tecnológica**. 2011. Dissertação (Pós-graduação em Engenharia Mecânica)-Universidade Federal de Uberlândia. Uberlândia, 2011. Disponível em: <[http://www.bdtu.ufu.br/tde\\_busca/arquivo.php?codArquivo=4123](http://www.bdtu.ufu.br/tde_busca/arquivo.php?codArquivo=4123)>. Acesso em: 18 mar. 2014.

MULTILÓGICA SHOP. **Módulo Bluetooth - BlueSMiRF HID**. [2013?]. Disponível em: <<http://multilogica-shop.com/m%C3%B3dulo-bluetooth-bluesmirf-hid>>. Acesso em: 18 abr. 2014.

MARQUES, Milena R. Sena. **Estudo das Linguagens para Modelagens de Sistemas Embarcados**. Universidade Federal de Pelotas. Pelotas, 2012. Disponível em: <<http://www.set.ufpa.br/setmine/attachments/253/Estudo-das-Linguagens-para-Modelagem-de-Sistemas-Embarcados.pdf>> Acesso em: 05 jun. 2014.

MINGUES, Altineu Pires. **Navegação: A Ciência e a Arte**. Volume III - NAVEGAÇÃO ELETRÔNICA E EM CONDIÇÕES ESPECIAIS. 2000. Disponível em: <[https://www.mar.mil.br/dhn/bhmn/publica\\_manualnav3.html](https://www.mar.mil.br/dhn/bhmn/publica_manualnav3.html)>. Acesso em: 13 maio 2013.

PRASAD, Mayank. **Sensores Fundamentais**. 2013. Disponível em: <<http://maxembedded.com/2011/06/18/sensor-fundamentals/>>. Acesso em: 18 abr. 2014.