

Universidade Federal de Itajubá *Campus* Itabira

Engenharia da Computação

Luiz Otávio Nunes Moura da Rosa

Rogério de Oliveira Batista

SISTEMA SUPERVISÓRIO DE TEMPO REAL: UMA IMPLEMENTAÇÃO DE BAIXO CUSTO

Itabira

2015

Luiz Otávio Nunes Moura da Rosa
Rogério de Oliveira Batista

SISTEMA SUPERVISÓRIO DE TEMPO REAL: UMA IMPLEMENTAÇÃO DE BAIXO CUSTO

Monografia parcial apresentada à Coordenação do Trabalho Final de Graduação, como requisito parcial, para obtenção do título de bacharel em Engenharia de Computação da Universidade Federal de Itajubá – *Campus Itabira*.

Universidade Federal de Itajubá *Campus Itabira*

Orientador: Prof. MSc. Walter Aoiama Nagai

Itabira

2015

Lista de ilustrações

Figura 1	– Estrutura básica de um sistema supervisorio.	14
Figura 2	– Arquitetura de Von-Neumman.	16
Figura 3	– Arquitetura Harvard apresentada em um microcontrolador PIC (a numeração representa a quantidade de <i>bits</i> do barramento).	17
Figura 4	– Diagrama de blocos simplificado de um Arduino.	18
Figura 5	– Visão de um sistema operacional.	20
Figura 6	– Técnicas de I/O.	23
Figura 7	– Escalonamento circular.	27
Figura 8	– Escalonamento circular com prioridades.	28
Figura 9	– Escalonamento por múltiplas filas.	29
Figura 10	– Tabela de alocação de partições.	30
Figura 11	– Processos de aplicação, <i>sockets</i> e protocolos de transporte subjacente. .	31
Figura 12	– Comportamento de requisição-resposta do HTTP.	32
Figura 13	– Cabeçalho de uma mensagem de requisição HTTP.	32
Figura 14	– Cabeçalho de uma mensagem de resposta HTTP.	33
Figura 15	– Representação da tradução de <i>bytecodes</i> para chamadas do sistema operacional.	34
Figura 16	– Estrutura de um código HTML.	36
Figura 17	– Métodos de um Servlet HTTP.	37
Figura 18	– Métodos de um Servlet HTTP.	38
Figura 19	– Estrutura de um código JSP.	38
Figura 20	– Estrutura de um JavaBean.	39
Figura 21	– Propriedades de um JavaBean.	39
Figura 22	– Modelo MVC.	40
Figura 23	– Modelo Cascata Puro utilizado no desenvolvimento de softwares	41
Figura 24	– Interação entre o banco de dados e o cliente.	46
Figura 25	– Interface da Eclipse IDE.	48
Figura 26	– Topologia do projeto.	53
Figura 27	– Arduino DUE.	54

Lista de tabelas

Tabela 1 – Recursos	57
Tabela 2 – Cronograma	59

Lista de abreviaturas e siglas

A/D	<i>Analógico/Digital</i>
ADC	<i>Conversor Analógico/Digital</i>
API	<i>Application Programming Interface</i>
ARM	<i>Advanced RISC Machine</i>
CISC	<i>Complex Instruction Set Computer</i>
CPU	<i>Central Processing Unit</i>
CSS	<i>Cascading Style Sheets</i>
D/A	<i>Digital/Analógico</i>
DAC	<i>Conversor Digital/Analógico</i>
DMA	<i>Direct Memory Access</i>
DHTML	<i>Dynamic Hypertext Markup Language</i>
DSP	<i>Digital Signal Processor</i>
EJB	<i>Enterprise Java Bean</i>
FIFO	<i>First In First Out</i>
FPGA	<i>Field Programmable Gate Array</i>
LCD	<i>Liquid Crystal Display</i>
HTML	<i>Hypertext Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IBM	<i>International Business Machines</i>
IDE	<i>Integrated Development Environment</i>
IHC	<i>Interface Homem Maquina</i>
I/O	<i>In/Out</i>
IP	<i>Internet Protocol</i>

Java EE	<i>Java Enterprise Edition</i>
JDBC	<i>Java Database Connectivity</i>
JDK	<i>Java Development Kit</i>
J2EE	<i>Java 2 Enterprise Edition</i>
JPA	<i>Java Persistence API</i>
JRE	<i>Java Runtime Environment</i>
JSF	<i>Java Server Faces</i>
JSP	<i>Java Server Pages</i>
JVM	<i>Java Virtual Machine</i>
MVC	<i>Model View Controller</i>
PIC	<i>Peripheral Interface Controller</i>
PDV	<i>Ponto de Venda</i>
PWM	<i>(Pulse-Width Modulation</i>
RAM	<i>Random Access Memory</i>
RISC	<i>Reduced Instruction Set Computer,</i>
RTO	<i>Real Time Operating System</i>
SCADA	<i>Supervisory Control and Data Acquisition</i>
SJF	<i>Shortest Job First</i>
SQL	<i>Structured Query Language</i>
SO	<i>Sistema Operacional</i>
TCP	<i>Transmission Control Protocol</i>
UDP	<i>User Datagram Protocol</i>
UML	<i>Unified Modeling Language</i>
URL	<i>Uniform Resource Locator</i>
WTP	<i>Web Tools Platforms</i>
XHTML	<i>Extensible Hypertext Markup Language</i>
XML	<i>Extensible Markup Language</i>

Sumário

1	INTRODUÇÃO	9
2	OBJETIVOS	11
2.1	Objetivo Geral	11
2.2	Objetivos Específicos	11
3	FUNDAMENTAÇÃO TEÓRICA	13
3.1	Estação Remota - Possibilidades de implementação	15
3.1.1	Microcontroladores <i>PIC</i>	16
3.1.2	Plataforma Arduino	17
3.1.3	Microcontroladores <i>Freescale</i>	18
3.1.4	Microcontroladores Texas Instruments	19
3.2	Sistemas Operacionais de Tempo Real, Interrupções e I/O Programada	20
3.3	Gerenciamento de memória e escalonamento de processos	24
3.3.1	Escalação de processos	24
3.3.2	Gerenciamento de memória	29
3.4	Redes de comunicação - Implementação e protocolos de funcionamento	30
3.5	A linguagem <i>Java</i>	33
3.6	A linguagem <i>HTML</i>	36
3.7	<i>Servlets</i>	36
3.8	Tecnologias <i>JSP</i> e <i>JavaBeans</i>	38
3.9	Estrutura <i>Model View Controller</i>	40
3.10	Estação de monitoramento - <i>Software</i> de gerenciamento de dados e tomada de decisões	41
3.11	Banco de dados e <i>driver JDBC</i>	44
3.12	A linguagem <i>CSS</i>	46
3.13	Ferramentas de Desenvolvimento	47
3.13.1	Ferramenta de desenvolvimento - <i>Eclipse IDE</i>	47
3.13.2	Servidor <i>Web</i> - <i>Apache Tomcat</i>	48
3.13.3	Sistema de Gerenciamento de banco de dados - <i>PostgreSQL</i>	48
3.13.4	Ferramenta de gerenciamento do <i>PostgreSQL</i> - <i>pgAdmin</i>	49
4	METODOLOGIA	51

5	DESENVOLVIMENTO	53
5.1	<i>Arduino DUE</i>	54
6	RECURSOS	57
7	CRONOGRAMA	59
8	RESULTADOS ESPERADOS	61
	Referências	63

1 Introdução

A busca por maneiras mais eficientes¹ de produção e gerenciamento no ambiente empresarial sempre foi um esforço constante. Mais do que uma questão de aumento de eficiência, no cenário atual, isso se tornou uma questão de sobrevivência no mercado. Um dos itens chave na busca do aumento da eficiência no meio industrial e comercial é a automatização de processos.

O sucesso de uma instituição/empresa está diretamente relacionado a forma com que o administrador/empresário otimiza os processos nela envolvidos. De maneira geral tem-se a seguinte situação: quanto maior o nível de automatização, maior será o seu faturamento bem como a suas margens de lucro. Além disso, o seu nível de profissionalização tende a se elevar. A partir da década de 70 vários setores da economia viram-se obrigados a seguir essa linha de pensamento, realizando a implantação em massa de técnicas de otimização em seus processos produtivos. De acordo com os dados da empresa de associação comercial de varejo **National Retail Federation** divulgados por (BEITOL, 2015), as lojas que utilizaram equipamentos e *softwares* com o intuito de automatizar a captura de dados no **Ponto de Venda (PDV)**, obtiveram em média 16% de aumento de vendas, com algumas lojas atingindo até 23%.

Processos que são operados e/ou controlados exclusivamente por funcionários, estão sujeitos a vários riscos e inconvenientes como por exemplo: erros provenientes de falha humana, informações insuficientes e/ou inconsistentes à respeito do processo de produção, dificuldade ou mesmo a impossibilidade de realização de um *feedback* confiável, queda de produtividade e consequentemente de lucros, entre outros.

A engenharia de controle ao lado da computação tiveram papel fundamental na solução desse problema, sendo responsáveis pelo surgimento dos conceitos de automação industrial e informatização pela primeira vez. Nos setores primários e secundários da economia, prevalece o conceito de automação industrial, que consiste em sistemas automáticos de *hardware* e/ou *software*. Já as soluções de automatização por *software* são amplamente utilizadas no setor terciário. Independentemente de qual setor da economia que a instituição se enquadre, o objetivo de tais sistemas é justamente eliminar ou pelo menos amenizar os problemas citados acima.

No meio industrial prevalece a utilização de computadores de pequeno porte, que tem como características o seu baixo poder de processamento (no entanto, capacidade suficiente para a execução da tarefa desejada) e baixo custo. Tais computadores são mais

¹ Compreende-se como aumento de eficiência todos os procedimentos que contribuam para o aumento de lucro da empresa.

conhecidos como microcontroladores, que são compostos por uma unidade central de processamento, memória, e pinos de I/O. Muitos modelos de microcontroladores vão além dessa estrutura básica, apresentando uma série de periféricos integrados. Entretanto, o processo de implantação de alguma solução de automação ou informatização, requer um estudo detalhado de como projetar, avaliar e adquirir os componentes necessários a implementação do sistema automatizado (sistemas automatizados podem ser constituídos tanto por soluções de *hardware*, *software* ou de ambas).

A utilização de microcontroladores em conjunto com algum *software* que tenha como objetivo a coleta, análise e armazenagem de dados faz surgir o conceito de sistema supervisorio. Em outras palavras, tal sistema consiste em uma ferramenta capaz de adquirir e armazenar informações, além de poder controlar uma grande variedade de processos.

O presente trabalho irá abordar a implementação de um sistema supervisorio de tempo real, destinado ao controle do fluxo de clientes em uma rede de motéis. A problemática do trabalho foi construída a partir de uma situação indesejada enfrentada por um empresário proprietário de uma rede de motéis na qual, o controle de entrada e saída de clientes é totalmente realizado de forma manual. O empresário passava por vários problemas oriundos dessa abordagem como por exemplo: falta de controle sob a entrada e saída clientes, o que provocava a geração de um fluxo de caixa inconsistente, além de extravios de dinheiro por parte dos funcionários responsáveis pelo controle do processo. Consultando várias soluções oferecidas pelo mercado, o empresário sentiu-se desmotivado em função do elevado custo relacionado a aquisição e implantação do sistema. Diante de tal situação, é formulada a seguinte problemática para o trabalho em questão: “Como implementar um sistema supervisorio de tempo real de baixo custo para realizar o controle do fluxo de clientes na rede de motéis? ”

O sistema a ser desenvolvido terá a sua parte de aquisição de dados constituída por um computador de baixa capacidade e sensores de proximidade que serão responsáveis pela detecção do fluxo de entrada e saída de clientes. A parte de armazenamento de informações será implementada por meio de um banco de dados online. A manipulação e análise dos dados será realizada por um *software* a ser desenvolvido que permitirá a visualização e interpretação dos dados adquiridos em tempo real.

2 Objetivos

Esta seção é direcionada a apresentação do objetivo geral e dos objetivos específicos do presente trabalho.

2.1 Objetivo Geral

O objetivo geral deste trabalho consiste no desenvolvimento de um sistema de supervisão e gerenciamento em tempo real para o controle de fluxo de clientes em um motel.

2.2 Objetivos Específicos

- Monitorar o fluxo de clientes de uma empresa em tempo real;
- Desenvolver um sistema para monitoramento de eventos com auxílio de sensores e um microcontrolador;
- Verificar a necessidade do uso de interrupções para gerenciar eventos;
- Criar um servidor para armazenar e tratar dados;
- Projetar sistema de envio de dados do microcontrolador, via interface **Ethernet**, para o servidor **Web**;
- Implementar banco de dados do sistema utilizando a linguagem **SQL**;
- Projetar uma interface **Web** em **Java** para gerenciamento dos eventos.

3 Fundamentação Teórica

Dando continuidade à discussão sobre sistemas autônomos nos processos produtivos, pode-se citar que são vários os benefícios conquistados pelas empresas e por seus clientes ao se utilizar essa abordagem:

- Redução de custos: Sistemas automatizados são capazes de reduzir custos de maneira significativa a curto prazo por meio do aumento de produtividade e eficiência;
- Qualidade: Máquinas autônomas fornecem resultados consistentes e repetíveis, sendo que, os problemas de controle de qualidade envolvidos como o erro humano são totalmente eliminados. Além disso, os processos podem ser cuidadosamente regulados e controlados de acordo com a necessidade do empresário;
- Segurança: Como já apresentado, sistemas automatizados estão praticamente imunes aos erros que um operador humano possa vir a cometer, caso estivesse realizando o seu trabalho. Sistemas automatizados industriais devem possuir a premissa de segurança fornecidas pelas normas regulamentadoras, que estão em vigor no país em questão;
- Monitoramento remoto: Sistemas de automação e de informatização oferecem a vantagem de permitir ao operador monitorar e/ou controlar os processos de produção fora do seu local de implantação. Dependendo da estrutura adotada e da implementação, esse monitoramento pode ser feito em tempo real.

Conforme apresenta (BARRETTO; MIYAGI; SILVA, 1993, p. 87), um sistema supervisório é constituído por uma plataforma computacional, que integra a análise de diversas transações de troca de informação e sinais entre um sistema de computação e um processo, de modo a obter um diagnóstico da evolução dos estados do sistema, atuando também sobre estes dados nos casos onde isto se faça necessário. Um modelo básico de um sistema supervisório é representado de acordo com a Figura 1.

O sistema apresentado na Figura 1 é também conhecido como SCADA (**Sistemas de Controle de Supervisão e Aquisição de Dados**) sendo que, a sua implementação irá depender da arquitetura de redes industriais utilizada. Na Figura 1, a estação de trabalho é responsável por armazenar o *software* de gerenciamento do SCADA, a rede de comunicações pode ser implementada de várias maneiras como: cabeamento **ethernet**, **bluetooth**, rádio frequência, entre outras. A estação remota é geralmente composta por algum microcontrolador que recebe sinais provenientes dos sensores e envia comandos aos atuadores oriundos da estação de monitoramento. Nesse ponto é importante que fique

Figura 1 – Estrutura básica de um sistema supervisorio.



Fonte: Autoria própria.

claro a diferença entre microprocessadores e microcontroladores. Uma definição bastante interessante é apresentada por (OLIVEIRA; ANDRADE, 2014) em que afirma que, microcontroladores assim como microprocessadores são circuitos integrados disponíveis nos mais variados tipos de encapsulamento, sendo destinados ao tratamento de sinais digitais. Entretanto, microcontroladores geralmente possuem todos os periféricos necessários num único *chip*. Microprocessadores por sua vez não conseguem realizar o seu trabalho sozinho, já que para o seu funcionamento é necessário a integração de outros dispositivos externos (periféricos) para que ele se torne útil.

Em muitos casos é desejável que o sistema apresentado na Figura 1 detecte os eventos assim que ocorrerem, ou seja, de maneira praticamente instantânea. Para que isso seja possível é necessário a utilização de um sistema de *software* que tenha suporte à execução de múltiplas tarefas em que o tempo de resposta a um evento seja conhecido. Independentemente do contexto, esse tempo pré-definido (conhecido também como tempo de resposta) deverá ser cumprido, ou seja, deverá ser determinístico. O não cumprimento desse prazo é tido como uma falha de sistema. Sistemas que apresentam essas características são denominados sistemas de tempo real, mais conhecidos pela sigla **RTOS** – **Real Time Operating System**. A detecção de eventos nesse tipo de sistema é baseada no uso de interrupções de *hardware*¹, sendo que o tratamento de tais interrupções irá depender do microcontrolador em questão. Esse tipo de sistema é utilizado em vários ramos da indústria, agricultura e comércio. Sua utilização está envolvida em sistemas dos mais variados níveis de complexidade e periculosidade: em coisas relativamente simples como jogos eletrônicos e videoconferências até a sistemas críticos de controle de usinas nucleares e freios automotivos. Mais informações a respeito de um **RTOS** são apresentadas nas próximas seções.

Outro aspecto importante na implementação do sistema da Figura 1 se diz respeito a arquitetura a ser adotada na estação remota. Nesse contexto é importante se ter em mente os conceitos de processadores **RISC** e processadores **CISC**. Basicamente processado-

¹ Um sinal que geralmente resulta em uma troca de contextos no processador.

res RISC (Reduce Instruction Set Computer) apresentam um conjunto de instruções reduzido enquanto processadores CISC possuem um conjunto complexo de centenas de instruções. Conforme apresenta (MARIMOTO, 2007, p. 1) a arquitetura CISC tem as seguintes vantagens:

“[...] um processador RISC é capaz de executar tais instruções muito mais rapidamente. A ideia principal, é que apesar de um processador CISC ser capaz de executar centenas de instruções diferentes, apenas algumas são usadas frequentemente. [...] é indiscutível, porém, que em instruções complexas os processadores CISC saem-se melhor.”

A vantagem da arquitetura CISC está relacionada ao seu vasto conjunto de instruções armazenadas no processador, o que facilita o trabalho dos programadores, que dispõem de praticamente todas as instruções que seus *softwares* farão uso. No caso de processadores RISC, *softwares* mais complexos teriam que combinar uma série de instruções a fim de executar a mesma tarefa mais complexa que poderia ser executada por apenas uma instrução CISC. Isso gera um maior nível de complexidade para o programador que está desenvolvendo a aplicação. Geralmente a estação remota é constituída por uma unidade RISC em função de suas operações na maioria dos casos consistirem por tarefas relativamente simples.

3.1 Estação Remota - Possibilidades de implementação

A estação remota é implementa utilizando-se algum tipo de microcontrolador que atenda às necessidades do projeto. Existem várias possibilidades quanto a modelos e marcas que podem ser utilizadas, sendo os tipos mais utilizados: Plataforma **Arduino**, famílias **PIC**, **FPGA**, famílias **Texas Instruments**, famílias **Freescall**. Independente da marca e família adotada no projeto, é importante ter em mente que, o microcontrolador deverá apresentar algumas características globalmente desejáveis que são:

- Baixo consumo energético;
- Baixo tempo de resposta;
- Baixo custo de aquisição;
- Alta confiabilidade;
- Imunidade a ruídos e interferências;

A seguir será apresentada uma breve discussão a respeito de cada um desses segmentos de microcontroladores.

3.1.1 Microcontroladores PIC

Trata-se de uma série de circuitos integrados produzidos pela empresa **Microchip Technology Inc.**, que apresentam integrado em um único dispositivo todos os circuitos necessários ao desenvolvimento de um sistema digital programável completo. A principal vantagem do PIC em relação ao restante dos microcontroladores é devido ao fato de fazer uso da arquitetura **Harvard** que, conforme apresenta (OLIVEIRA; ANDRADE, 2014, p. 27):

“Na arquitetura de **Von Neumann** existe apenas um barramento por onde passam os dados de memória de programa e da memória de dados, enquanto na arquitetura de **Harvard** existem barramentos diferentes, proporcionando maior desempenho, pois permitem múltiplos acessos à memória, ou seja, acessos simultâneos à memória de dados e à memória de programa.”

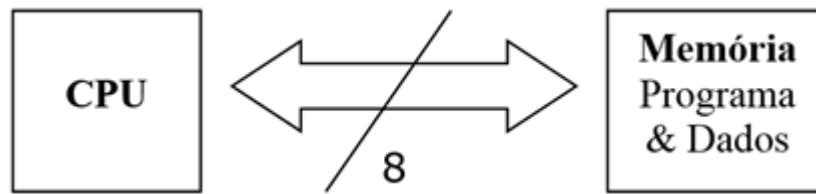
Os microcontroladores com arquitetura **Havard** são também conhecidos como “microcontroladores **RISC**” enquanto os microcontroladores com arquitetura **Von Neumann**, de “microcontroladores **CISC**”. O alto desempenho da família de microcontroladores PIC pode ser atribuído as seguintes características proporcionadas pela arquitetura **RISC**:

- Mapa de Registradores versátil;
- Todas as instruções com palavras simples;
- Palavra de instrução longa;
- Arquitetura de instruções em *pipeline*;
- Instruções de apenas um cliço de máquina;
- Conjunto de instruções reduzido;
- Conjunto de instruções ortogonal (simétrico).

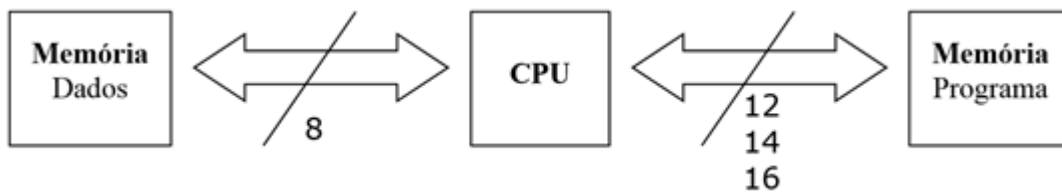
A Figura 2 e Figura 3 apresentam as diferenças entre as arquiteturas de **Von-Neumman** e **Havard** respectivamente:

Existem várias versões de **chips** na família PIC, tendo modelos capazes de processar dados de 8, 12, 16 ou 32 *bits*. Esses dispositivos são amplamente utilizados na indústria quanto por “*hobystas*” em função do seu baixo custo, ampla disponibilidade e grande acervo de material de desenvolvimento. Com relação ao seu desempenho, os microcontroladores PIC tem a sua arquitetura implementada de forma a maximizar a relação de velocidade-custo. O seu conjunto de instruções se mostra adequado a construção de tabelas de consulta rápida, sendo que, tais consultas ocupam uma instrução e dois ciclos de

Figura 2 – Arquitetura de Von-Neumman.



Fonte: Autoria própria.

Figura 3 – Arquitetura Harvard apresentada em um microcontrolador PIC (a numeração representa a quantidade de *bits* do barramento).

Fonte: Autoria própria.

instrução. Apesar disso, a plataforma PIC está perdendo espaço para outras alternativas presentes no mercado. Isso se deve ao fato do alto custo de seu gravador (desestimulando a sua utilização em pequenos projetos) e poder de processamento insatisfatório nas versões de 8 *bits* quando comparado a outros dispositivos que também trabalham com esse tamanho de palavra.

3.1.2 Plataforma Arduino

Arduino consiste em uma plataforma de prototipagem aberta que utiliza microcontroladores ATMEL AVR. Consiste em uma solução relativamente recente no mercado, tendo sido criada em 2005 com o objetivo de servir como base para projetos de baixo custo. Utiliza o mesmo conjunto de instruções reduzido de instruções (RISC) utilizado nos dispositivos PIC. Sua principal característica é a simplicidade, que faz com que seja bastante popular no meio dos desenvolvedores amadores e experientes; sua utilização não requer conhecimentos avançados de eletrônica.

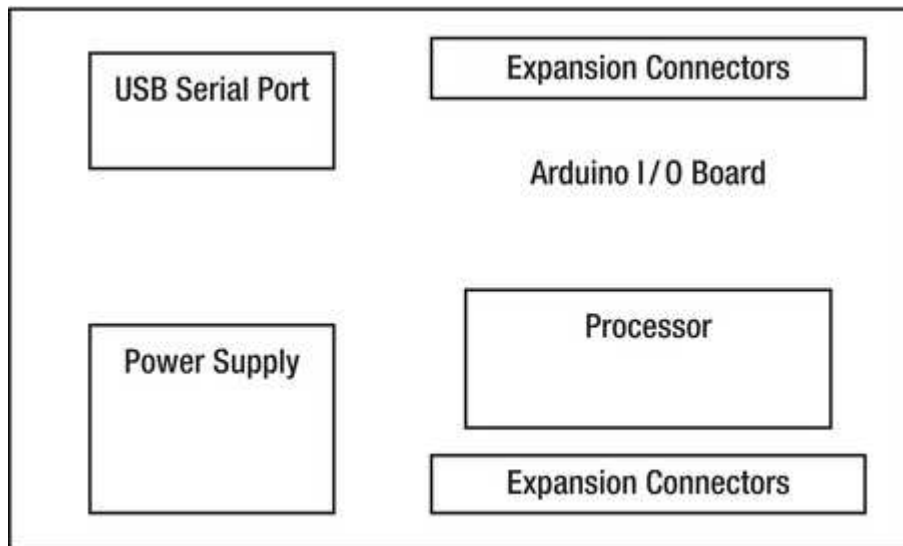
Essa ideia é reforçada por (VASILJEVIC, 2013, p. 3) apresentando que:

“O **Arduino** foi criado com o propósito de ser uma plataforma extremamente fácil de usar se comparado às outras [...] Outro fator que torna o **Arduino** atrativo é sua filosofia de *hardware* livre, ou seja, pessoas podem usá-lo para criar diversos projetos sem custo algum de direitos pela utilização da plataforma [...]”

Dentre as soluções apresentadas para o desenvolvimento de uma estação remota de um sistema SCADA, o **Arduino** é a que apresenta a documentação mais diversificada

e acessível, contando com milhares de comunidades na internet de pessoas que divulgam informações e detalhes dos projetos que criam. O campo de aplicações de tal plataforma é ilimitado. A Figura 4 apresenta o diagrama de blocos simplificado de um **Arduino**.

Figura 4 – Diagrama de blocos simplificado de um **Arduino**.



Fonte: (WHEAT, 2011, p. 2).

Existem diferentes versões de **Arduino**, como a **UNO**, **MEGA**, **LEONARDO**, **DUE**, **MEGA ADK**, **NANO**, **MINI** e **ESPLORA**. A versão a ser adotada irá depender das características do projeto. A grande maioria de suas versões são baseadas no chip **ATmega8** e seus derivados. Informações detalhadas a respeito de cada versão podem ser encontradas no livro "**Arduino Internals**" escrito por Dale Wheat. O núcleo do **Arduino** processa dados de 8 ou 32 *bits*, sendo esta última versão destinada a projetos que requerem maior poder de processamento. A versão **DUE** é a que apresenta maior poder de processamento sendo baseada em um núcleo **ARM Cortex-M3** que funciona até 84 *MHz*. As demais versões de 8 *bits* trabalham com um *clock* de até 16 *MHz*.

3.1.3 Microcontroladores *Freesc*ale

A fabricante **Freesc**ale Semiconductor, Inc. possui uma diversificada linha de microcontroladores, sendo o seu foco a área de sistemas embarcados e comunicação. Sua linha atual de projetos são todos baseados na arquitetura **ARM**. Atualmente a empresa tem o seu foco na família de microcontroladores **Kinetis** e na plataforma de desenvolvimento **Freedom** que utiliza os núcleos dessa família. As placas **Freedom** são equipadas com microcontroladores **Kinetis** e são uma forma rápida, prática e barata para os primeiros contatos com os dispositivos desenvolvidos pela empresa. A placa já vem com alguns dispositivos integrados como *touch pad*, sensor infravermelho, termistor, conectores compatíveis com *shield Arduino R3*, entre outros. Por se tratar de uma plataforma de desenvolvimento

com vários periféricos integrados, apesar do seu baixo custo, sua utilização pode não ser viável em alguns tipos de projeto. A linha **Freedom** é baseada no núcleo ARM **Cortex-M0+** ou no ARM **Cortex-M4**, conforme cita (LIMA, 2013) em seu site, tais unidades baseadas no **Cortex-M0+** implementam a arquitetura de Von Neumann sendo microcontroladores de 32 *bits* de baixo custo desenvolvidos pela ARM, que tem por objetivo ocupar o lugar de microcomputadores bem simples que realizam funções específicas ou que demandam baixo consumo de energia. O núcleo funciona com uma velocidade de *clock* de até 48 MHz. Entre as novidades apresentadas pela linha **Freedom**, destacam-se a implementação de dois estágios de *pipeline* o que diminui os acessos feitos pelo processador à memória *flash*, consequentemente, diminuindo o consumo da unidade central. As interrupções possuem uma latência menor em relação a outros núcleos, proporcionando um tempo de chamada de função bem pequeno o que o torna bastante interessante a aplicações em RTOS. Além disso foi implementado um sistema de unidade protegida de memória em que há suporte a execução em nível de privilégio ou não privilegiado. As unidades baseadas no núcleo **Cortex-M4** além das características já citadas, apresentam maior custo e poder de computação, podendo operar a até 120 Mhz. Além disso, as unidades **Cortex-M4** apresentam maiores quantidades de memória RAM e memória *flash*.

3.1.4 Microcontroladores Texas Instruments

A Texas Instruments é uma empresa que atua na área de semicondutores oferecendo as mais variadas soluções na área de microcontroladores DSPs, conversores A/D e D/A. Dentre as centenas de modelos oferecidos pela empresa, atualmente a sua melhor relação custo-benefício para implementação de sistemas supervisórios são os microcontroladores da linha MSP430 já que reúne praticidade e simplicidade de aplicação com um custo acessível. Conforme apresenta o (ELETRÔNICA, 2011) essa nova família apresenta baixo consumo de corrente, combinado com periféricos inteligentes e integrados, sendo que o MSP430 é ideal para aplicações de baixo custo, incluindo equipamentos industriais e médicos portáteis, medição inteligente e aparelhos eletrônicos de consumo, assim como uma ampla gama de novas aplicações. Esse dispositivo utiliza a arquitetura RISC de 16 *bits*. Os membros dessa família diferem entre si em questões relacionadas à quantidade memória *flash* e memória RAM além dos periféricos integrados disponíveis. A CPU MSP430 vem equipada com um conjunto de apenas 51 instruções (sendo 27 físicas e 24 emuladas) e um total de 16 registradores de 16 *bits*. Essa CPU trabalha com um *clock* de até 16 MHz e atualmente é uma forte concorrente da plataforma **Arduino**.

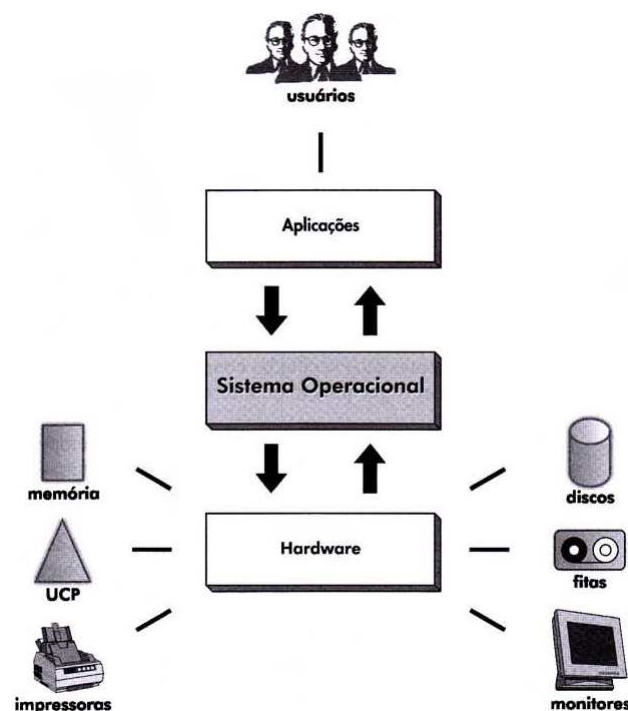
3.2 Sistemas Operacionais de Tempo Real, Interrupções e I/O Programada

Conforme apresenta (LIMA; VILLAÇA, 2012, p. 473) um sistema operacional – SO é definido como sendo:

“[...] uma coleção de programas que atua como uma interface entre programas do usuário e o *hardware*. Sua principal função é proporcionar um ambiente para que o usuário de um sistema microprocessado execute programas no *hardware* do sistema de uma maneira conveniente e eficiente.”

O SO é o responsável pelo gerenciamento do processador, da memória e dos dispositivos de entrada e saída do sistema. A partir disso é fácil concluir-se a grande complexidade envolvida em sua implementação; isso vale mesmo em SO's relativamente simples como por exemplo os embarcados em fornos micro-ondas, máquinas de lavar e robôs que se quer possuem interface gráfica. A diferença básica entre um sistema operacional e uma aplicação convencional está na maneira em que as rotinas são executadas. Sistemas operacionais tem suas rotinas executadas concorrentemente em função de eventos assíncronos. A Figura 5 apresenta a visão geral de um sistema operacional e o seu posicionamento nessa topologia.

Figura 5 – Visão de um sistema operacional.



Fonte: (MACHADO; MAIA, 2007, p. 4).

Analisando-se a definição proposta por (LIMA; VILLAÇA, 2012) no início do primeiro parágrafo desta seção, entende-se como conveniente o fato citado por (MACHADO; MAIA, 2007) em que o sistema operacional é tido como uma espécie de interface entre o usuário e os recursos disponíveis no sistema computacional responsável por tornar esta comunicação transparente.

Como apresentado anteriormente, um sistema RTOS é um sistema operacional que proporciona um comportamento temporal previsível. Dessa forma idealmente um RTOS garante que a grande maioria de suas tarefas serão executadas em um prazo de tempo constante, independentemente da carga de trabalho a qual a CPU estará exposta naquele momento. Sistemas operacionais tradicionais como Windows 8.1 e Linux não apresentam essa característica (determinismo), ao contrário de um RTOS, apenas garantem que as tarefas serão executadas, porém, o tempo de execução não é preciso sendo muitas vezes referido como "tempo médio de execução". Um RTOS é um sistema dirigido a eventos de I/O. Em função disso, a priori é necessário a apresentação das principais técnicas utilizadas no processo de transferência de dados entre o processador e a memória principal antes de dar continuidade a uma discussão mais aprofundada a respeito de um sistema RTOS.

Existem alguns tipos de aplicação que por mais que necessitem das características fornecidas por um RTOS, nem sempre será necessário à sua utilização. Isso é possível por meio da utilização do conceito de interrupções.

Sistemas embarcados utilizados em microcontroladores não costumam utilizar nenhuma espécie de sistema operacional, toda a sua programação é desenvolvida utilizando-se o conceito de *superloop* que consiste em um único laço que é executado de maneira sequencial. A execução do *superloop* é realizada indefinidamente enquanto a placa permanecer energizada. Nesse tipo de abordagem, eventos críticos (leitura de um sensor por exemplo) são tratados através de interrupções de *hardware* que são executadas em primeiro plano. As demais operações do laço principal são executadas em segundo plano. A partir do instante em que a CPU do microcontrolador recebe um sinal de interrupção, inicia-se a chamada rotina de tratamento de interrupção. Essa rotina é caracterizada pela troca de contexto em que, o contexto atual (valores de registradores, contador de programa, etc.) é salvo na memória RAM e o tratamento da interrupção (rotina de *software* específica) é iniciada. Após o término do tratamento da rotina, os registradores são restaurados e o controle é transferido para a instrução seguinte ao ponto de interrupção da aplicação. No caso de sistemas de tempo real pode ser que algumas aplicações críticas não possam ser interrompidas durante a sua execução. Nesse contexto começa a surgir o conceito de prioridade de interrupções, que é mais conhecido como interrupções mascaráveis e não-mascaráveis.

Interrupções mascaráveis podem ser desabilitadas pelo processador, enquanto as não-mascaráveis não, ou seja, deverão ser sempre atendidas. Pode ser que outras inter-

rupções surjam enquanto o processador está executando uma rotina de tratamento, nesse caso, tem-se o surgimento dos vetores de interrupção que são armazenados em uma tabela na memória principal.

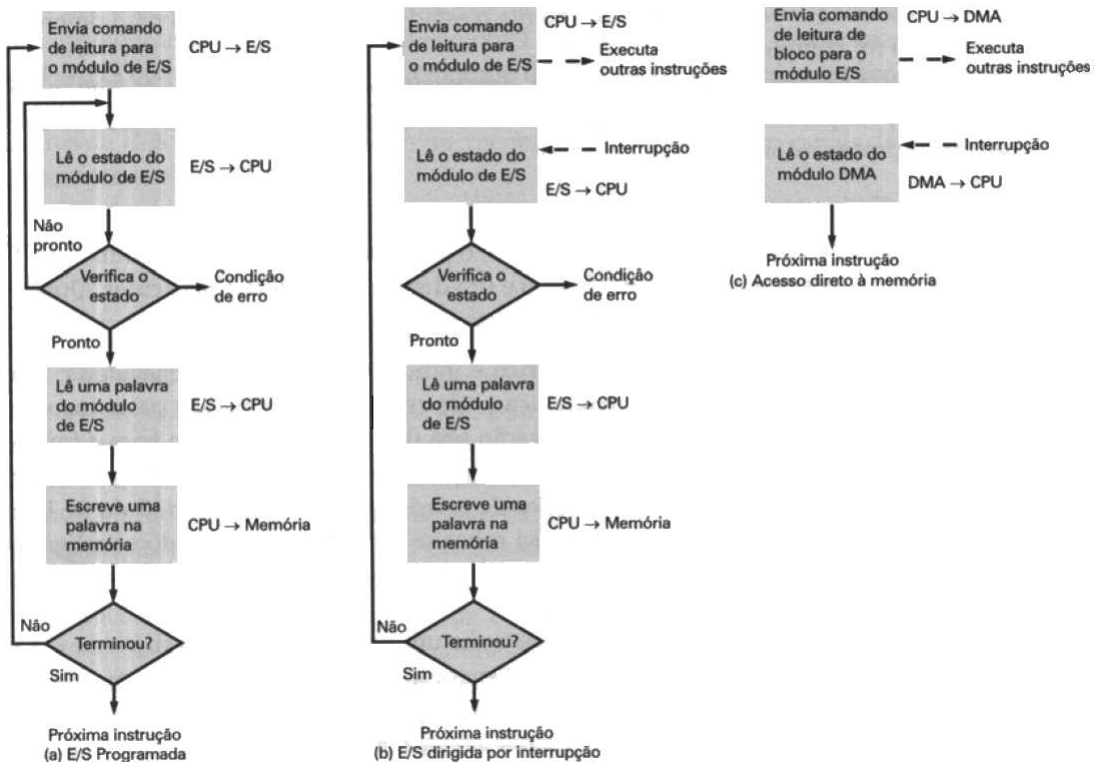
Porém, a abordagem comumente utilizada na programação de microcontroladores seja por questões de projeto ou por inexperiência do programador é a chamada de I/O programada conhecida também como “espera ocupada”. A utilização de interrupções permite que o processador execute outras tarefas na ausência de um sinal de interrupção. É como se o processador fosse “avisado” pelo dispositivo (sensor) o instante em que o evento ocorre. Já na I/O programada, conforme apresenta (STALLINGS, 2003) o processador fica responsável pelo controle direto da operação de I/O, incluindo a detecção do estado do dispositivo, o envio de comandos de leitura ou escrita e a transferência de dados. É de se esperar que haverá um grande desperdício de uso de CPU caso o módulo de I/O seja mais lento que ela (o que ocorre na maioria das vezes). Em outras palavras, essa técnica faz com que o processador fique checando constantemente o estado do módulo I/O a fim de verificar o seu estado. Além do desperdício de tempo de processamento, o uso de I/O programada pode gerar inconsistências na detecção de eventos em códigos mais extensos; como o código é executado de maneira sequencial pode ser que, no instante de ativação de um sensor por exemplo, a execução esteja em um ponto em que isso não é tratado. Isso é muito comum em sistemas de monitoramento que são sistemas que podem ter que lidar com um grande número de sensores.

Tanto no caso do uso de interrupções como na “espera ocupada” a CPU sempre é a responsável por intermediar a comunicação entre o módulo I/O e à memória principal. Existe uma técnica alternativa conhecida como DMA (**D**irect **M**emory **A**ccess) em que a transferência entre o módulo de I/O e a memória principal é feita diretamente sem a intervenção do processador. Ainda segundo (STALLINGS, 2003), as duas primeiras formas de I/O apresentadas anteriormente possuem duas desvantagens inerentes:

- Taxa de transferência limitada pela velocidade com que o processador pode testar e servir um dispositivo;
- Ocupação do processador no gerenciamento da transferência, tendo que executar várias instruções a cada transferência.

A grande sacada da técnica DMA consiste na presença de um módulo DMA capaz de simular o trabalho executado pelo processador, ou seja, tal módulo fica responsável por todo o gerenciamento da operação de entrada e saída entre o dispositivo e a memória principal. A utilização do barramento pelo módulo DMA pode ser realizado por meio de duas maneiras diferentes: o módulo força a CPU a liberar o barramento ou o usa quando a CPU não está o ocupando. A Figura 6 apresenta as características de cada uma das três técnicas de comunicação discutidas.

Figura 6 – Técnicas de I/O.



Fonte: (MACHADO; MAIA, 2007, p. 4).

Apesar das grandes vantagens associadas ao uso da técnica DMA, apenas microcontroladores mais robustos (destinados ao tratamento de arquivos de áudio MP3, *streaming* de vídeo, etc.) possuem suporte a tal tecnologia. Entre os modelos de aplicação geral que possuem um módulo DMA estão a família de chips AVR XMEGA da Atmel e PIC32 do microchip, além da plataforma Arduino DUE equipada com chip ARM.

A estratégia de utilização de interrupções sem a utilização de um SO é eficaz até certo ponto. A medida que o número de interrupções cresce o código tende a se tornar mais complexo e ineficiente em função do crescimento do vetor de interrupções². Nesse cenário pode ocorrer situações indesejáveis como estouro da pilha de execução, monopólio da CPU por um processo, entre outras. Dessa forma é fácil perceber que, independentemente da técnica de I/O utilizada pelo programador, sem a utilização de um sistema operacional fica a seu cargo realizar todo o gerenciamento dos recursos. Esse gerenciamento envolve o controle de: memória e processos. O gerenciamento dessas duas entidades é algo complexo de se implementar (principalmente no que diz respeito ao gerenciamento de memória). Mais detalhes a respeito do gerenciamento de memória e de processos são discutidos na seção próxima seção.

² Tabela de endereços de memória que apontam para as rotinas de tratamento de interrupção.

Sistemas operacionais como um todo apresentam a grande vantagem de serem transparentes ao usuário no que diz respeito a gerência de processos e memória de maneira bem mais eficiente e segura. No caso dos microcontroladores equipados com um RTOS temos além disso, a capacidade de transformar o conceito de *superloop* em uma aplicação baseada em múltiplas tarefas que são adequadamente gerenciadas pelo **kernel** do sistema.

3.3 Gerenciamento de memória e escalonamento de processos

Nesta seção são apresentadas as principais atribuições de um sistema operacional que a grosso modo se resumem ao gerenciamento de memória e o escalonamento de processos.

3.3.1 Escalonamento de processos

O gerenciamento da CPU consiste em uma das atividades mais importantes envolvidas em um sistema de computação. Deve-se ter políticas apropriadas responsáveis por determinar qual processo será o escolhido para fazer o uso da unidade central de processamento. Tais políticas são conhecidas como políticas de escalonamento. A parte do SO responsável por implementar os critérios da política de escalonamento é conhecida como escalonador. Independentemente da política a ser utilizada durante o escalonamento de processos é objetivo geral do escalonador manter a CPU a maior parte do tempo ocupada e balancear o seu uso entre os processos de acordo com a sua prioridade. Além disso, conforme apresenta (MACHADO; MAIA, 2007) existem alguns critérios que devem ser levados em conta em qualquer política de escalonamento, são eles:

- Utilização do processador: Garante que o processador irá permanecer ocupado na maior parte do tempo;
- Throughput: Representa o número de processos executados em um determinado intervalo de tempo. Quanto maior o **throughput**, maior o número de tarefas executadas em função do tempo. A maximização do **throughput** é desejada na maioria dos sistemas;
- Tempo de Processador: Se refere ao tempo que um processo leva no estado de execução durante seu processamento. As políticas de escalonamento não influenciam o tempo de um processo, sendo este tempo função apenas do código da aplicação e da entrada de dados;
- Tempo de Espera: Se refere ao tempo total que um processo permanece na fila de pronto durante seu processamento, aguardando para ser executado. A redução de

tempo de espera dos processos é desejada pela maioria das políticas de escalonamento;

- **Tempo de Resposta:** Se refere ao tempo decorrido entre uma requisição ao sistema ou à aplicação e o instante em que a resposta é exibida. Em geral, o tempo de resposta não é limitado pela capacidade de processamento do sistema computacional, mas pela velocidade dos dispositivos de I/O.

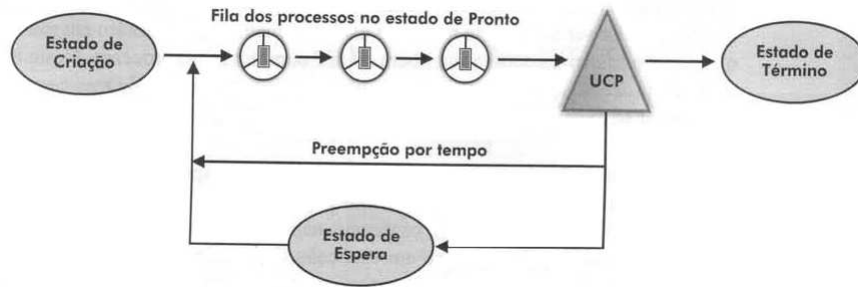
Em linhas gerais (MACHADO; MAIA, 2007) afirmam que, “qualquer política de escalonamento busca otimizar a utilização do processador e o **throughput**, enquanto tenta diminuir os tempos de **turnaround**, espera e resposta”.

São várias as técnicas utilizadas na operação de escalonamento de processos, algumas mais sofisticadas, outras nem tanto. O mesmo pode-se dizer em relação ao quesito complexidade, alguns algoritmos são mais eficientes, porém, complexos e vice e versa. O fato é que, não existe nenhuma solução ideal e que seja a prova de falhas. Os algoritmos escalonadores se dividem em dois grandes grupos, os preemptivos e os não preemptivos. O primeiro caso consiste nos algoritmos que permitem que um processo seja interrompido (por meio de um sinal de interrupção) durante a sua execução. Essa abordagem dá ao sistema a possibilidade de priorizar a execução de processos, como ocorre no caso de aplicações de tempo real. O último caso é basicamente o contrário do primeiro, como o próprio nome indica, ou seja, não existe a possibilidade de nenhum evento externo interromper a execução do processo que está fazendo uso da CPU no momento. As únicas possibilidades de o processo sair do estado de execução são: caso termine seu processamento ou execute uma instrução própria que faça isso. Algoritmos não preemptivos não são condizentes com as propostas de um RTOS ou de qualquer sistema em que múltiplos processos requerem a sua atenção. A seguir são apresentados resumidamente alguns algoritmos populares que podem ser utilizados na implementação do escalonador. A descrição de cada técnica foi adaptada (MACHADO; MAIA, 2007, p. 137-150).

- **Escalonamento First-In-First-Out (FIFO):** Nessa técnica, o processo que chegar primeiro ao estado de pronto é o selecionado para execução. Sua implementação exige apenas a criação de uma fila na qual os processos em estado pronto entrem no seu final e são escalonados quando chegam ao seu início. Quando o processo em execução termina seu processamento ou vai para o estado de espera, o primeiro processo da fila de pronto é escalonado. Apesar de simples, essa técnica apresenta um grave problema que consiste na impossibilidade de se prever quando um processo terá a sua execução iniciada. Além disso vale ressaltar que é um algoritmo não preemptivo;

- Escalonamento **Shortest-Job-First (SJF)**: O algoritmo seleciona o processo em estado de pronto que necessitar de menos tempo de CPU. Como não é possível precisar previamente o tempo de processador para cada processo, uma estimativa é realizada utilizando-se análises estatísticas de execuções passadas dos programas. Caso o tempo informado ao **SJ** for muito inferior ao tempo real, o sistema poderá interromper a execução do processo. O principal problema nesta implementação é a impossibilidade de estimar o tempo de processador para processos interativos, devido à entrada de dados ser uma ação imprevisível;
- Escalonamento Cooperativo: Busca aumentar o grau de multiprogramação em políticas de escalonamentos que não possuam mecanismos de preempção, como o **FIFO** e o **SJF** não-preemptivo. Neste caso, um processo em execução pode voluntariamente liberar o processador retornando a fila de pronto. Sua principal característica está no fato de a liberação do processador ser uma tarefa realizada exclusivamente pelo processo em execução, ficando a seu cargo a verificação periódica de uma fila de mensagens para determinar se existem outros processos na fila de pronto. No caso de um processo não verificar a fila de mensagens, os demais não terão chance de serem executados até a liberação do processador pelo processo em execução;
- Escalonamento Circular: É um escalonamento do tipo preemptivo, projetado especialmente para sistemas de tempo compartilhado. É muito semelhante ao **FIFO**, porém, quando um processo passa para o estado de execução, existe um tempo limite para o uso contínuo do processador denominado fatia de tempo ou quantum. O algoritmo concede uma nova fatia de tempo para cada processo toda vez que ele é escalonado para execução. A Figura 7 ilustra o funcionamento do algoritmo. O valor da fatia de tempo varia de acordo da arquitetura de cada sistema operacional e este valor está diretamente relacionado ao desempenho da política de escalonamento circular. No caso em que a fatia de tempo tenha um valor muito alto, o escalonamento circular irá apresentar o mesmo comportamento do escalonamento **FIFO**. Sua principal vantagem é a não permissão da monopolização da CPU por um processo, já que a sua fatia de tempo é definida pelo sistema. Por outro lado, processos do tipo **CPU-bound** são beneficiados no uso do processador em relação a processos **I/O-bound** ocasionando um balanceamento desigual do processador entre os processos;
- Escalonamento por Prioridades: É um escalonamento preemptivo realizado com base em um valor associado a prioridade de execução de cada processo. Processos com maiores prioridades são sempre os primeiros a serem escolhidos para execução, processos com prioridades iguais são escolhidos de acordo com o critério **FIFO**. Como não existe o conceito de fatia de tempo, um processo em execução não pode sofrer preempção por tempo. Dessa forma a perda do uso do processador só ocorrerá no caso de uma mudança voluntária para o estado de espera ou quando o processo passa

Figura 7 – Escalonamento circular.

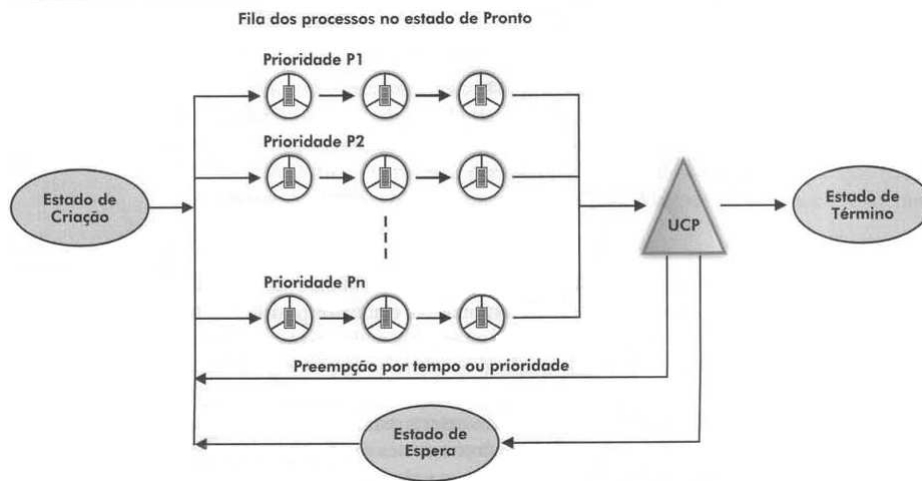


Fonte: (MACHADO; MAIA, 2007, p. 142).

para o estado de pronto. A implementação dessa lógica é realizada através de uma interrupção de *clock*, sendo que, no caso do surgimento de um processo de maior prioridade na fila de pronto, o SO realiza a preempção. Esse tipo de escalonamento também pode ser alternativamente implementado de maneira não-preemptiva, neste caso um processo não perderá o uso de CPU mesmo que um de maior prioridade surja na fila de prontos (o processo será apenas alocado para o início da fila). Os valores de prioridades constituem uma característica de *software* e podem ser fixos ou dinâmicos. Um dos principais problemas no escalonamento por prioridades é o **starvation**. Processos de baixa prioridade podem não ser escalonados, permanecendo indefinidamente na fila de pronto. Isso pode ser resolvido por meio da aplicação da técnica conhecida como **aging**, que consiste em incrementar gradualmente a prioridade de processos que permanecem por muito tempo na fila de pronto;

- **Escalonamento Circular com Prioridades:** Neste tipo de escalonamento existe o conceito de fatia de tempo aliada ao conceito de prioridade. Um processo permanece em execução até que termine seu processamento, ou sofra uma preempção por tempo. A Figura 8 ilustra o seu funcionamento. Sua principal vantagem é permitir o melhor balanceamento no uso da CPU, com a possibilidade de diferenciar o grau de importância dos processos;
- **Escalonamento por Múltiplas Filas:** Nesta metodologia existem diversas filas de processos no estado de pronto, cada qual com uma prioridade específica. Cada fila tem características próprias, como importância para a aplicação, tipo de processamento ou área de memória necessária. Em função de cada processo possuir características distintas, torna-se difícil que um único mecanismo de escalonamento seja adequado a todos. Aí que reside a principal vantagem do escalonamento por múltiplas filas já que é possível a convivência de escalonamento distintos (FIFO ou circular) em cada fila em um mesmo SO. Não existe prioridade, isso se torna uma característica associada à fila. O processo em execução sofre preempção caso outro processo entre

Figura 8 – Escalonamento circular com prioridades.



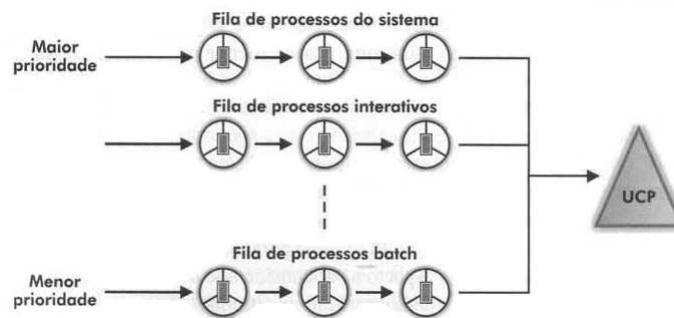
Fonte: (MACHADO; MAIA, 2007, p. 146).

em uma fila de maior prioridade. O SO só poderá escalonar um processo de determinada fila caso todas as filas de maior prioridade estejam vazias. A Figura 9 ilustra a técnica de escalonamento por múltiplas filas;

- Escalonamento por Múltiplas Filas com Realimentação: É semelhante ao escalonamento por múltiplas filas, porém os processos podem trocar de fila durante seu processamento. A vantagem disso é que o SO pode identificar dinamicamente o comportamento de cada processo, direcionando-o para a fila com prioridade de execução e mecanismo de escalonamento mais adequados ao longo do seu processamento. Um mecanismo FIFO adaptado com fatia de tempo é implementado para o escalonamento em todas as filas, com exceção da fila de menor prioridade que continua utilizando o escalonador circular. A fatia de tempo destinada a cada processo varia de acordo com a sua prioridade. Este tipo de escalonamento atende as necessidades dos diversos tipos de processos. No caso de processos I/O-bound, um tempo de resposta adequado é obtido, já que esses processos têm prioridades mais altas por permanecerem a maior parte do tempo nas filas de maior prioridade, em que dificilmente sofrerão preempção por tempo.

Em linhas gerais vale a pena destacar que sistemas com prioridade dinâmica são mais complexos de serem implementados, porém, apresentam um menor tempo de resposta. Segundo (MACHADO; MAIA, 2007) atualmente a maioria dos SOs de tempo compartilhado utiliza o escalonamento circular com prioridades dinâmicas. No caso de sistemas de tempo real, a metodologia mais indicada é a pôr prioridades. Nesse tipo de sistema não deverá haver o conceito de fatia de tempo e a prioridade de cada processo deverá ser estática.

Figura 9 – Escalonamento por múltiplas filas.



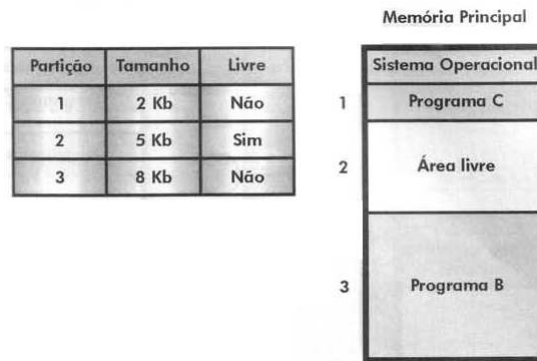
Fonte: (MACHADO; MAIA, 2007, p. 147).

3.3.2 Gerenciamento de memória

Em sistemas multiprogramados a memória é dividida basicamente em duas partes, a memória de usuário e a memória do sistema. A área de memória do usuário é subdividida de maneira que acomode vários processos. Essa subdivisão é feita pelo SO e ocorre dinamicamente, sendo tal procedimento conhecido como gerenciamento de memória. Assim como no gerenciamento de processos, o gerenciamento de memória é vital para um sistema multiprogramado. Deve-se alocar a memória de maneira eficiente e com a maior quantidade possível de processos. Os primeiros sistemas operacionais utilizavam técnicas de particionamento estáticas em que, o tamanho de cada partição é fixo. O controle é realizado por meio de uma tabela que contém o endereço inicial e final de cada partição, além do seu tamanho e se a mesma está ou não em uso. Quando um programa precisa ser carregado na memória, o SO realiza uma busca na tabela a fim de encontrar uma partição livre onde o programa possa ser carregado. O grande problema da alocação fixa é o surgimento do fenômeno conhecido como fragmentação interna de memória. Isso ocorre porque nem sempre os programas preenchem totalmente as partições onde são carregados. A Figura 10 apresenta a estrutura de uma tabela de partições. Para solucionar-se o problema da segmentação interna foi criado um método de alocação dinâmica em que o conceito de partições fixas foi eliminado. Cada programa utiliza apenas o espaço que necessita, o problema da fragmentação interna é resolvido. Porém, a partição dinâmica produz um novo tipo de segmentação, conhecida como segmentação externa.

Isso ocorre na medida em que a memória é utilizada e a entrada e saída de processos vai produzindo espaços vazios cada vez menores ao longo do espaço de endereçamento. Nesse contexto a memória poderá até ter o espaço suficientemente livre para que um determinado processo seja alocado, entretanto, geralmente tal espaço não está disposto de maneira contínua o que impede a sua alocação. A tendência é que com o passar do tempo a memória se torne cada vez mais fragmentada e a sua utilização diminua. Existe uma técnica apresentada por (STALLINGS, 2003) que acaba em parte com esse problema; denominada compactação consiste em uma varredura que é executada de tempos em

Figura 10 – Tabela de alocação de partições.



Fonte: (MACHADO; MAIA, 2007, p. 161).

tempos pelo SO com o objetivo de trocar os processos de lugar, agrupando dessa forma todas as áreas livres de memória. A desvantagem é que esse processo consome tempo útil de processamento. Conforme apresenta (MACHADO; MAIA, 2007), existem basicamente três estratégias de alocação de espaços na memória principal, são elas:

- **Best-fit:** Essa técnica visa escolher a melhor partição, ou seja, a que o programa deixa o menor espaço sem utilização. Entretanto, a sua grande desvantagem é a sua tendência de fazer com que a memória fique cada vez mais com áreas não-contíguas, aumentando o problema da fragmentação;
- **Worst-fit:** O conceito dessa técnica é o inverso da anterior, ou seja, é escolhida a partição que deixa o maior espaço sem utilização. Isso é interessante pois maiores partições surgirão o que permite um número maior de programas utilizando a memória, amenizando o problema da fragmentação;
- **First-fit:** É escolhida a primeira partição livre de tamanho suficiente para carregar o programa. A lista de áreas livres é ordenada de maneira crescente por endereços. Em função de utilizar as áreas livres de endereços mais baixos, existe uma grande chance de se obter uma grande partição livre nos endereços de memória mais altos. É a estratégia mais rápida, consumindo menos recursos do sistema.

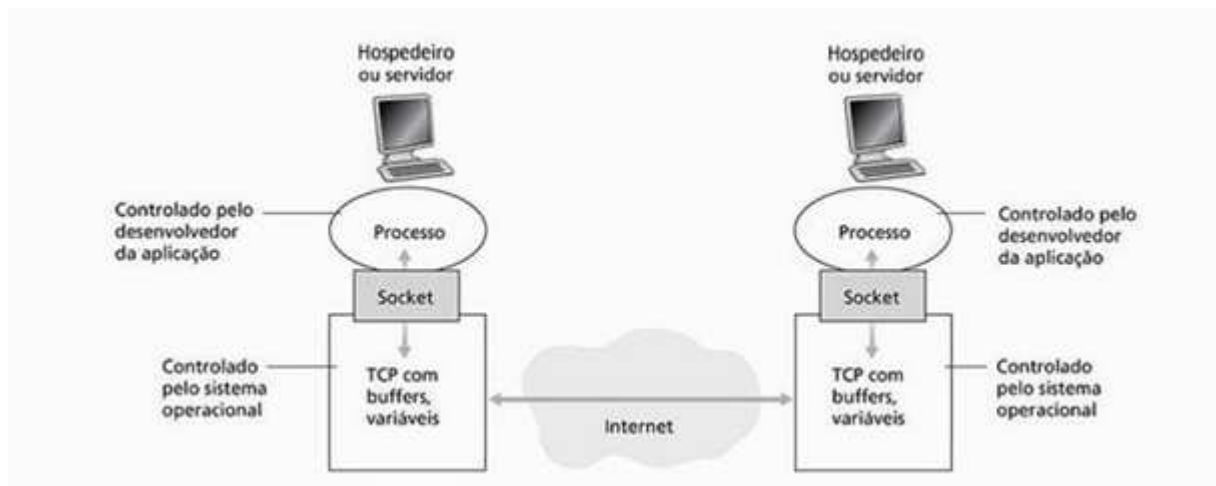
3.4 Redes de comunicação - Implementação e protocolos de funcionamento

A comunicação entre o microcontrolador e o servidor *Web* será realizada de acordo com os padrões da comunicação cliente-servidor, onde o microcontrolador fará o papel de cliente e a interface *Web* juntamente com o banco de dados será o servidor. A comunicação entre o cliente e o servidor será realizada através do protocolo HTTP (**HyperText**

Transfer Protocol) que faz parte da camada de aplicação da *Web*. O programa inserido no microcontrolador conversará com o programa criado no servidor através da troca de mensagens HTTP. O HTTP utiliza o TCP como protocolo de transmissão de dados, essa operação é descrita por (KUROSE; ROSS, 2010, p. 73) da seguinte forma:

“O cliente HTTP primeiramente inicia uma conexão TCP com o servidor. Uma vez estabelecida a conexão, os processos do browser e do servidor acessam o TCP por meio de suas interfaces *sockets*. O cliente envia mensagens de requisição HTTP para sua interface *socket* e recebe mensagens de resposta HTTP de sua interface *socket*. De maneira semelhante, o servidor HTTP recebe mensagens de requisição de sua interface *socket* e envia mensagens de resposta para sua interface *socket*. Assim o cliente envia uma mensagem para sua interface *socket*, a mensagem sai de suas mãos e “passa para as mãos do TCP” [...]. O TCP prove ao HTTP um serviço confiável de transferência de dados, o que implica que toda mensagem de requisição HTTP emitida por um processo cliente chegara intacta ao servidor. De maneira semelhante, toda mensagem de resposta HTTP emitida pelo processo servidor chegará intacta ao cliente [...].”

Figura 11 – Processos de aplicação, *sockets* e protocolos de transporte subjacente.



Fonte: (KUROSE; ROSS, 2010, p. 66).

Existem dois tipos de mensagens HTTP conforme explicado por (KUROSE; ROSS, 2010, p. 76), as mensagens de requisição e as mensagens de resposta. As mensagens de requisição possuem um cabeçalho com as informações da mensagem. Uma linha deste cabeçalho diz respeito ao método que a mensagem de requisição carrega. Os métodos existentes são *GET*, *POST* e *HEAD*. O método *GET* é normalmente o mais utilizado, ele consiste na requisição de um objeto feita a um servidor. O método *HEAD* é semelhante ao *GET*, porém este método não será abordado neste projeto. O método *POST* é utilizado, por exemplo, quando um usuário preenche um formulário e envia as alterações para o servidor. Segundo descreve (KUROSE; ROSS, 2010, p. 77):

”Com uma mensagem *POST*, o usuário continua solicitando uma página *Web* ao servidor, mas o conteúdo específico dela depende do que o usuário

escreveu nos campos do formulário. Se o valor do campo de método for *POST*, então o corpo de entidade conterá o que o usuário digitou nos campos do formulário.“

A mensagem de resposta é utilizada quando o servidor envia uma mensagem para o cliente com o intuito de responder a uma mensagem de requisição.

Figura 12 – Comportamento de requisição-resposta do HTTP.



Fonte: (KUROSE; ROSS, 2010, p. 73).

O formato das mensagens de requisição é apresentado na Figura 13.

Figura 13 – Cabeçalho de uma mensagem de requisição HTTP.

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr
```

Fonte: (KUROSE; ROSS, 2010, p. 76).

O significado de cada linha é apresentado por (KUROSE; ROSS, 2010, p. 76). A primeira linha é composta pelo método utilizado de conexão, o endereço da requisição ou destino da mensagem e a versão do HTTP. A segunda linha apresenta o endereço de onde foi feita a requisição, ou seja, a origem. Na terceira linha o campo “*Connection*” representa o tipo de conexão utilizado. Os tipos de conexões existentes são classificados como persistentes ou não persistentes. Em conexões persistentes o cliente estabelece uma conexão com o servidor permanecendo, conectado enquanto houver a transferência de dados ou o cliente desejar. Já em conexões não persistentes, os clientes e servidores estabelecem uma conexão apenas pelo período necessário ao envio de uma mensagem. Neste caso o campo “*Connection: Close*” estabelece uma conexão não persistente, onde o cliente fica conectado ao servidor somente até a resposta da mensagem de requisição. A quarta linha apresenta o tipo do *browser* que está fazendo a requisição ao servidor. A quinta linha especifica a preferência de linguagem que o cliente deseja receber os dados, caso a

linguagem desejada não esteja disponível a mensagem de resposta conterá a linguagem padrão do servidor.

Já a mensagem de resposta é apresentada na Figura 14.

Figura 14 – Cabeçalho de uma mensagem de resposta HTTP.

```
HTTP/1.1 200 OK
Connection: close
Date: Sat, 07 Jul 2007 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 6 May 2007 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

Fonte: (KUROSE; ROSS, 2010, p. 78).

Conforme apresenta (KUROSE; ROSS, 2010, p. 78), a mensagem de resposta é dividida em três seções. Uma linha inicial ou de estado, seis linhas de cabeçalho e em seguida o corpo da mensagem. A primeira linha mostra a versão do HTTP seguido da confirmação de encontro do objeto requisitado. Na segunda linha é estabelecido se a conexão é persistente ou não. A terceira linha indica a data e hora em que o servidor enviou a mensagem de resposta. A quarta linha indica a versão de *software* que o servidor está utilizando, de forma análoga, esse campo é semelhante ao campo “*User-agent*” indicado na mensagem de requisição. A quinta linha indica a data e hora em que o objeto requisitado foi criado ou modificado pela última vez. Na sexta linha é apresentado o tamanho em *bytes* do objeto que está sendo enviado. A sétima linha mostra o formato do objeto a ser enviado. Por fim é apresentado o conteúdo de dados requisitado pelo cliente.

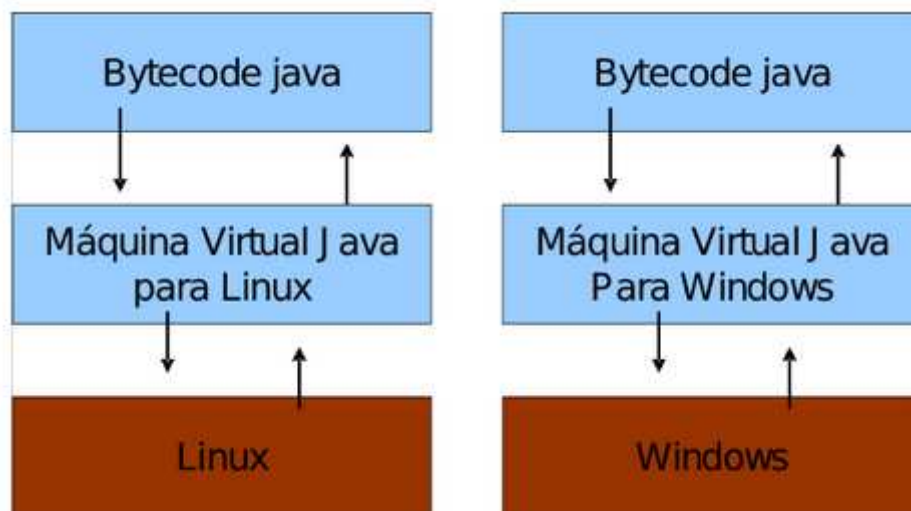
Considerando o projeto a ser desenvolvido os funcionários da empresa ou administradores farão papel de clientes no processo de gerenciamento. Eles irão fazer requisições de dados ao servidor e utilizar esses dados requisitados para otimizar o gerenciamento da empresa. Ao requisitar os dados do servidor os funcionários e/ou administradores estarão fazendo uso implícito do método *GET*. Já do lado dos microcontroladores que também são clientes, o método utilizado na comunicação HTTP será o *POST*, pois os microcontroladores apenas enviarão pulsos digitais ou informações que serão tratadas no servidor.

3.5 A linguagem Java

A linguagem Java foi criada pela antiga Sun Microsystems com objetivo sendo direcionado ao consumidor de produtos e usuários leigos. Em 1992 a Sun criou um time para desenvolver inovações tecnológicas. Esse time foi liderado por James Gosling, considerado pai do Java. James Gosling e seu time tinham como objetivo o desenvolvimento de um interpretador para pequenos dispositivos eletrônicos. A ideia no princípio fracassou devido a divergência de ideias e elevados custos. Com o crescimento da Web a Sun percebeu que poderia utilizar sua ideia para rodar pequenas aplicações dentro de um browser. Seu objetivo era que a linguagem Java rodasse nos diferentes tipos de browser e sistemas operacionais existentes na época. No ano de 2009, a Oracle comprou a Sun fortalecendo a marca. Desde então, a Oracle juntamente com a IBM foram as empresas que mais investiram na linguagem Java (CAELUM, a).

Para executar uma aplicação Java é necessário uma máquina virtual que interpreta e executa as operações encapsuladas em *bytecode* da linguagem. Essa máquina virtual recebe o nome de JVM (Java Virtual Machine). Esta característica importante da linguagem Java, permite que uma aplicação Java possa ser executada em qualquer sistema operacional, necessitando apenas da JVM (CAELUM, a).

Figura 15 – Representação da tradução de *bytecodes* para chamadas do sistema operacional.



Fonte: (CAELUM, a, p. 6).

Para desenvolver uma aplicação Java é necessário a utilização do JDK (Java Development Kit). O JDK é um pacote disponibilizado pela Oracle e é composto por um conjunto de utilitários necessários para criação e execução de aplicações em Java. Neste pacote está incluso um outro pacote chamado de JRE (Java Runtime Environment). O JRE é formado pela JVM e bibliotecas da linguagem.

Java é uma ferramenta poderosa para desenvolvimento de aplicações *Web*. Tal tecnologia possui um extenso conjunto de ferramentas e padronizações que são agrupadas em um pacote chamado de **Java Platform**, que visam economizar tempo e dinheiro no ciclo de desenvolvimento. Foi escolhida a linguagem **Java** para o desenvolvimento da aplicação *Web*, por ser uma linguagem orientada a objetos robusta de enorme versatilidade e importância no cenário atual da computação mundial.

Existe uma plataforma **Java** exclusiva para o desenvolvimento *Web*. Esta plataforma é chamada de **Java EE (Java Platform, Enterprise Edition)**. A plataforma **Java EE** possui bibliotecas e funcionalidades já criadas, baseadas em componentes modulares que são executados em servidores de aplicações. Esta plataforma é composta de uma série de tecnologias com objetivos distintos, as mais conhecidas são:

- **Servlets**: São códigos escritos em **Java** e são executados no servidor para gerar conteúdo dinâmico para *Web*;
- **JSP (JavaServer Pages)**: É uma especialização de um **Servlet** também utilizada para gerar conteúdo dinâmico;
- **JSF (JavaServer Faces)**: É um *framework Web* baseado em **Java**, foi criado com o intuito de facilitar a criação de telas (interface) e aumentar a produtividade no desenvolvimento de sistemas *Web*;
- **JPA (Java Persistence API)**: É uma API padrão do **Java** utilizada para persistência dos dados onde são utilizados conceitos de mapeamento objeto-relacional;
- **EJB (Enterprise Java Beans)**: São componentes que são executados no servidor com o objetivo de oferecer segurança e facilidade na produção de sistemas *Web*.

A aplicação **Java Web** a ser criada terá como principal objetivo receber e tratar os dados dos eventos reportados pelos serviços clientes, como o microcontrolador, conforme descreve (OLIVEIRA, 2004):

”Geralmente é bastante conveniente executar código de aplicação no servidor de banco de dados, ao invés de simplesmente recuperar dados executar a lógica da aplicação em processos separados. *Stored Procedures* possibilitam que a lógica da aplicação seja armazenada e executada no servidor.”

O desenvolvimento de *softwares* que utilizam o conceito de programação orientada a objetos trouxe inúmeros benefícios aos desenvolvedores de *software*, como o favorecimento do reuso ou extensão de classes criadas em outros projetos, conforme apresenta (SOARES et al., , p. 1). Normalmente os sistemas *Web* são distribuídos portanto, a orientação a objetos é um importante paradigma nesse contexto, simplificando o acesso

de diferentes usuários a quaisquer objetos. Em geral a programação orientada a objetos trouxe muitas facilidades no desenvolvimento de *softwares*, mas como qualquer outro estilo de programação ela possui seus limites e não elimina alguns problemas. Parte desses problemas são destacados por (RESENDE; SILVA, 2005, p. 16) como entrelaçamento, espalhamento, manutenibilidade e reusabilidade. Entretanto é importante ressaltar que o entrelaçamento consiste no acréscimo demasiado de código.

3.6 A linguagem HTML

A sigla HTML significa HyperText Markup Language, ou seja, linguagem de Marcação de Hipertexto. Consiste em uma linguagem de marcação utilizada para produção de páginas Web. Foi criada por Tim Barners na década de 1990.

A linguagem HTML permite a criação de páginas na *World Wide Web*, com imagens, textos, formulários, listas e conexões em hipertexto a outras páginas e arquivos. Para se escrever em HTML pode-se utilizar um editor de texto especial para linguagem ou um simples editor de texto, desde que sejam respeitados os comandos da linguagem. Os comandos HTML aparecem sempre na forma <comando>, escritos entre “<” e “>” para se diferenciarem do resto do texto em uma página. A Figura 16 mostra um simples exemplo criado na linguagem HTML.

Figura 16 – Estrutura de um código HTML.



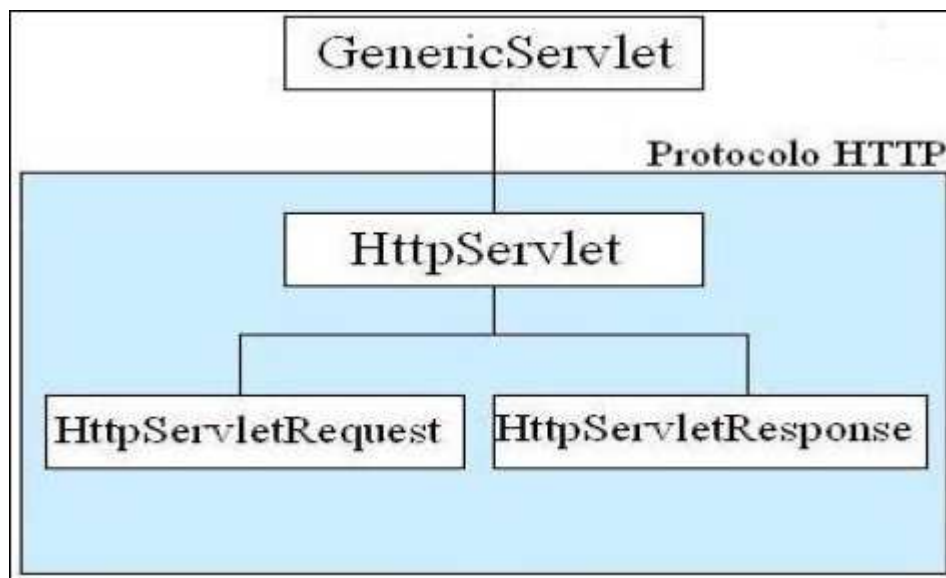
Fonte: Autoria própria.

3.7 Servlets

No princípio do surgimento da *Web* a maioria das páginas possuíam apenas conteúdos estáticos, compostos por códigos escritos no formato HTML. Esses códigos eram responsáveis por apresentar em sua maior parte textos, imagens, animações e outros conteúdos de caráter estático. Com o passar do tempo surgiu a necessidade da criação de páginas de conteúdo dinâmico, baseadas nas requisições em tempo real dos usuários. Em páginas com conteúdo dinâmico o usuário faz uma requisição ao servidor, que por sua vez processa essa requisição e retorna uma resposta. Na plataforma **Java** a principal tecnologia capaz de gerar páginas dinâmicas são as **Servlets** conforme apresenta (CAELUM, b, p. 52).

O nome **Servlet** é derivado do conceito de pequeno servidor (“servidorzinho” em inglês). O seu principal papel é o de responder a requisições HTTP. Em **Java**, um **Servlet** é representado como um objeto de uma classe **Java**. Os **Servlets** utilizam o mesmo modelo apresentado no protocolo HTTP. Este modelo é chamado de requisição/resposta. Cada **Servlet** é um objeto que possui como parâmetro dois campos: *request* (requisição) e *response* (resposta). Um **Servlet** pode responder a requisições dos mais diversos tipos de protocolo e não somente o HTTP (CAELUM, b, p. 53). Neste trabalho será abordado apenas as requisições e repostas com o protocolo HTTP .

Figura 17 – Métodos de um Servlet HTTP.



Fonte: (BARALE, 2007, p. 18).

A classe que possui as bibliotecas necessárias a implementação de um **Servlet** para o protocolo HTTP é chamada de **HTTP Servlet**. Esta classe possui uma série de métodos que estão listados na Figura 18

Os métodos mais utilizados são o "**doGet**" e "**doPost**" e serão os únicos que se-

Figura 18 – Métodos de um Servlet HTTP.

Modifier and Type	Method and Description
protected void	<code>doDelete(HttpServletRequest req, HttpServletResponse resp)</code> Called by the server (via the service method) to allow a servlet to handle a DELETE request.
protected void	<code>doGet(HttpServletRequest req, HttpServletResponse resp)</code> Called by the server (via the service method) to allow a servlet to handle a GET request.
protected void	<code>doHead(HttpServletRequest req, HttpServletResponse resp)</code> Receives an HTTP HEAD request from the protected service method and handles the request.
protected void	<code>doOptions(HttpServletRequest req, HttpServletResponse resp)</code> Called by the server (via the service method) to allow a servlet to handle a OPTIONS request.
protected void	<code>doPost(HttpServletRequest req, HttpServletResponse resp)</code> Called by the server (via the service method) to allow a servlet to handle a POST request.
protected void	<code>doPut(HttpServletRequest req, HttpServletResponse resp)</code> Called by the server (via the service method) to allow a servlet to handle a PUT request.
protected void	<code>doTrace(HttpServletRequest req, HttpServletResponse resp)</code> Called by the server (via the service method) to allow a servlet to handle a TRACE request.
protected long	<code>getLastModified(HttpServletRequest req)</code> Returns the time the <code>HttpServletRequest</code> object was last modified, in milliseconds since midnight January 1, 1970 GMT.
protected void	<code>service(HttpServletRequest req, HttpServletResponse resp)</code> Receives standard HTTP requests from the public service method and dispatches them to the <code>doXXX</code> methods defined in this class.
void	<code>service(ServletRequest req, ServletResponse res)</code> Dispatches client requests to the protected service method.

Fonte: (ORACLE, 2015).

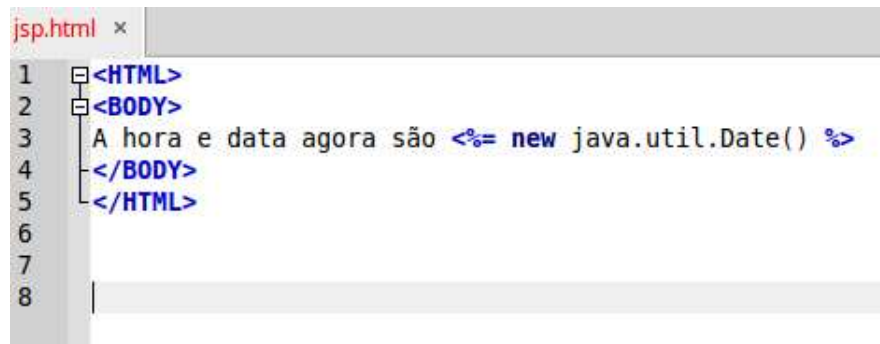
rão abordados neste trabalho. Eles funcionam de forma análoga aos métodos “*GET*” e “*POST*” empregados em conexões HTTP. Um **Servlet** possui um ciclo de vida dividido em três etapas: inicialização, execução e remoção. Primeiramente ele é carregado no servidor e permanece lá até que seja destruído. Após a inicialização, o **Servlet** estará apto a responder as requisições de usuários, etapa na qual caracteriza a sua execução. A etapa de remoção é utilizada para remover um **Servlet** do servidor, caso não seja removido ele permanecerá ocupando espaço e processamento no servidor (BARALE, 2007, p. 21).

3.8 Tecnologias *JSP* e *JavaBeans*

A tecnologia **Java Server Pages** permite gerar conteúdo da *Web dinâmico*, como arquivos DHTML, HTML, XHTML e XML, que farão parte de uma aplicação para *Web*. Uma página JSP consiste em uma codificação HTML mesclada com uma codificação Java inserida entre *TAGS*. JSP faz parte da plataforma J2EE (**Java 2 Enterprise Edition**) e em conjunto com os **Java Servlets** e **Java Beans** pode ser utilizado para desenvolver aplicações *Web* eficientes, escaláveis e seguras. Esta tecnologia é importante principalmente por permitir que o programador separe conteúdos lógicos de conteúdos visuais, oferecendo uma melhor dinâmica de desenvolvimento. Além disso, ela permite a interação com os mais diversos tipos de bancos de dados existentes graças a tecnologia JDBC (**Java Database Connectivity**) que será apresentada nos tópicos seguintes (JEVEAUX, 2008, p. 3). Na Figura 19 é apresentado um modelo minimalista de uma codificação na linguagem JSP.

O autor (JEVEAUX, 2008) descreve um **Bean** como uma classe em Java que segue um conjunto de convenções simples de *design* e nomeação determinado pela especifica-

Figura 19 – Estrutura de um código JSP.



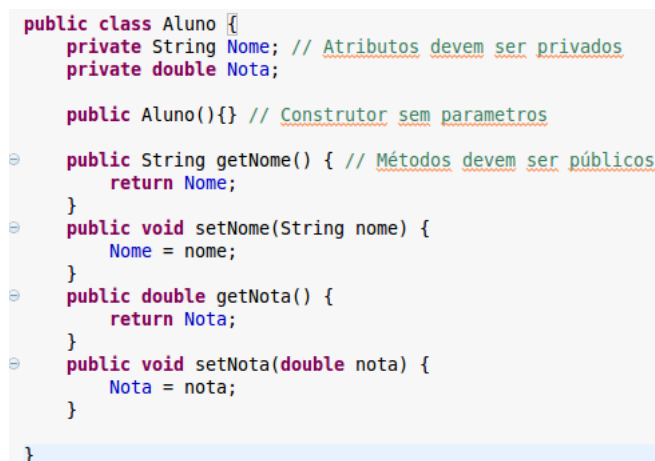
```
1 <HTML>
2 <BODY>
3 A hora e data agora são <%= new java.util.Date() %>
4 </BODY>
5 </HTML>
6
7
8
```

Fonte: Autoria própria.

ção JavaBeans. Para criar uma estrutura que obedece os conceitos de um `JavaBean` é necessário seguir os seguintes passos:

- O construtor de uma classe não deve receber argumentos;
- Podem existir diversos métodos públicos para determinar o valor de um atributo em um objeto. Esses métodos são chamados de métodos *setter*;
- Podem existir diversos métodos públicos para obter o valor de um atributo em um objeto. Esses métodos são chamados de métodos *getter*.

A Figura 20 apresenta como é implementada uma simples classe `JavaBean`.

Figura 20 – Estrutura de um `JavaBean`.

```
public class Aluno {
    private String Nome; // Atributos devem ser privados
    private double Nota;

    public Aluno(){} // Construtor sem parametros

    public String getNome() { // Métodos devem ser públicos
        return Nome;
    }
    public void setNome(String nome) {
        Nome = nome;
    }
    public double getNota() {
        return Nota;
    }
    public void setNota(double nota) {
        Nota = nota;
    }
}
```

Fonte: Autoria própria.

Por convenção e boa prática de programação, uma classe `JavaBean` precisa ter todos os seus atributos como privados e seus métodos “*getters*” e “*setters*” como públicos. Além disso, é aconselhável que os métodos comecem com a primeira letra minúscula e

colocar em maiúscula a primeira letra de cada palavra subsequente como apresenta a Figura 21 (JEVEAUX, 2008).

Figura 21 – Propriedades de um JavaBean.

```
private String corCarro;  
public String getCorCarro( ){  
    //Código  
}
```

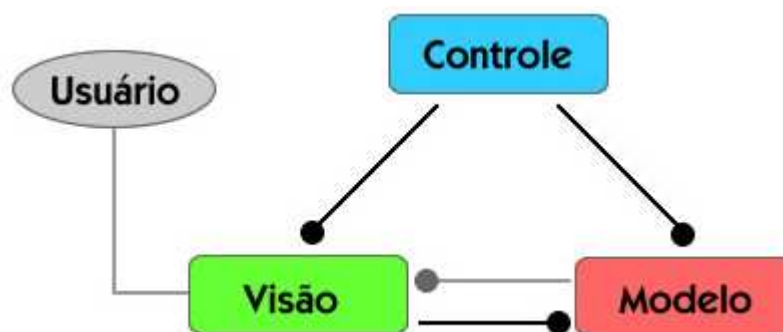
Fonte: Autoria própria.

É importante destacar aqui a diferença entre JavaBeans e EJB (Enterprise JavaBeans). A JavaBeans não tem o propósito de ser utilizado em componentes distribuídos como a EJB têm.

3.9 Estrutura Model View Controller

A arquitetura de um sistema normalmente é composta de diversos elementos tais como, elementos de interação, elementos de conexão, elementos de persistência entre outros. Em sistemas de *softwares* complexos a maneira como esses elementos se relacionam pode gerar dificuldades de implementação. Com o objetivo de padronizar a arquitetura de desenvolvimento de um *software*, foi criado o modelo Model View Controller (MVC). O modelo MVC visa dividir uma aplicação em três partes: o modelo, a visão e o controlador (MEDEIROS, 2014, p. 1).

Figura 22 – Modelo MVC.



Fonte: (MARTINS, 2011).

Explicando cada uma das partes do modelo MVC tem-se primeiramente o modelo (*Model*) que é responsável pelo gerenciamento e acesso aos dados que estão inclusos no banco de dados da aplicação. A visão (*View*) é responsável pela interação entre o usuário e aplicação, é nela que os dados serão apresentados ao usuário. O controlador (*Controller*) é responsável por delegar ao modelo as solicitações da visão. De forma geral a visão recebe

as informações do modelo, mas o modelo não tem conhecimento da visão. O responsável por essa comunicação é o controlador.

A principal ideia da arquitetura MVC é a separação dos códigos de forma organizada. O MVC é semelhante a clássica programação orientada a objetos e será de extrema importância neste trabalho. Entre as principais vantagens desta arquitetura é importante ressaltar a possibilidade de alteração de códigos contidos em qualquer uma das três partes sem gerar problemas nas partes adjacentes (MEDEIROS, 2014).

3.10 Estação de monitoramento - *Software de gerenciamento de dados e tomada de decisões*

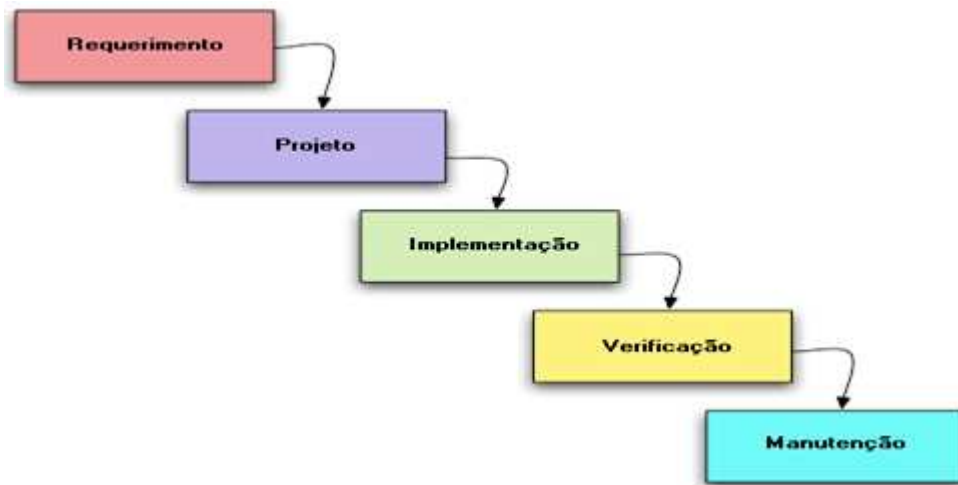
A engenharia de *software* terá um papel importante no desenvolvimento do sistema de gerenciamento de dados *Web*. Segundo (FALBO, 2005) a engenharia de *software* surgiu com o objetivo de melhorar a qualidade dos produtos de *software* e aumentar a produtividade no processo de desenvolvimento. A estratégia para aplicação dos conceitos de engenharia de *software* consiste na decomposição do problema original em partes menores. Esse método torna possível a solução das partes decompostas, para posterior integração das soluções.

Com a crescente competitividade no ambiente empresarial, a eficiência de um sistema está diretamente relacionada a sua qualidade. Segundo a NBR ISO 9000:2005, “qualidade é o grau no qual um conjunto de características inerentes satisfaz aos requisitos”. Para o *software* apresentar um bom grau de qualidade, é necessário que a suas funcionalidades atendam de forma eficaz todos os requisitos pré-estabelecidos.

Modelos podem ser utilizados para auxiliar a criação de um *software*. O autor (COLE, 2011) apresenta um exemplo de modelo cascata que foi uma das primeiras propostas para a decomposição das etapas de um projeto, almejando a apresentação do projeto em sequência. A simplicidade do modelo cascata facilita a explicação do projeto para pessoas que não estão familiarizadas com o desenvolvimento de *software*. A Figura 23 apresenta como funciona o modelo em cascata.

Ainda segundo (COLE, 2011), na fase de requerimentos são estabelecidos os requisitos do sistema a ser desenvolvido. Esta etapa também pode ser chamada de engenharia de requisitos, em que é feita a modelagem conceitual do sistema. A etapa de projeto tem como objetivo definir como serão atingidos os requerimentos estabelecidos, de modo que o conjunto de informações adquiridas permita o início da etapa de implementação. Na fase de implementação o programa começa a ser estruturado e codificado. A base de dados também é criada e todos os módulos que compõe o projeto são interligados. Após o término da fase de implementação é iniciada a etapa de verificação, que consiste na reali-

Figura 23 – Modelo Cascata Puro utilizado no desenvolvimento de **softwares**.



Fonte: (COLE, 2011).

zação de diversos testes sob o sistema projetado. Estes testes tem como objetivo encontrar erros e inconsistências na utilização do *software*. Com o *software* concluído e entregue a empresa, é iniciada a etapa de manutenção. A etapa de manutenção consiste em oferecer suporte a empresa caso problemas ou erros sejam encontrados pela empresa.

O desenvolvimento de sistemas que utilizam conceitos de orientação a objetos nas fases de análise de requisitos, análise de sistemas e *design* não oferecia uma notação padrão específica e eficaz que guiasse no projeto de tais sistemas. Existem diversas simbologias e cada uma com seus próprios conceitos. Visando resolver tal problema, foi criado por um grupo de amigos entusiastas da área de modelagem orientada a objetos, a "**Unified Modeling Language**" (UML). A UML é uma padronização de notação na parte de modelagem orientada a objetos. Essa padronização visava a modelagem de sistemas (não apenas de *software*) que utilizavam dos conceitos de orientação a objetos, a união de métodos conceituais com executáveis e a criação de uma linguagem de modelagem usável tanto pelo homem como por uma máquina. Seu principal objetivo é descrever qualquer sistema em termos de diagramas orientado a objetos (UML...,).

Ainda segundo a apostila (UML...,) que descreve a UML como a composição de 4 componentes que estão apresentados abaixo:

- Visões: A visão é um conjunto de diagramas que mostram os diferentes aspectos de um sistema que está sendo modelado;
- Modelos de elementos: Representam definições comuns da orientação a objetos como as classes, objetos, mensagem, relacionamento entre classes incluindo associações, dependências e heranças;

- Mecanismos gerais: Provém comentários suplementares, informações, ou semântica sobre os elementos que compõe o modelo;
- Diagramas: Os diagramas são os gráficos que descrevem o conteúdo em uma visão.

Os diagramas utilizados pela UML são compostos por nove tipos: diagramas de caso de uso, de classes, de objetos, de estado, de sequência, de colaboração, de atividade, de componente e o de execução.

O diagrama de casos de uso descreve o comportamento do sistema de uma forma geral. Nele são especificados argumentos funcionais que determinarão a arquitetura geral do sistema. Para a UML um requisito normalmente é uma funcionalidade desejada do sistema. O diagrama de casos de uso é responsável por detalhar os requisitos solicitados. Todas as funções específicas do sistema identificadas na fase de requisitos devem se tornar casos de uso.

O diagrama de classes é utilizado na modelagem estrutural de um sistema orientado a objetos. Este diagrama é considerado estático pois a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema. As classes possuem atributos, operações e associações entre outras classes. Todas estas peculiaridades estão inclusas em um diagrama de classe. Sistemas podem possuir um ou mais diagramas de classe, já que não são todas as classes que se relacionam.

O diagrama de objetos é utilizado para mapear a relação entre os objetos de cada classe. É muito semelhante ao diagrama de classes, porém é composto de instancias que possuem os valores dos atributos dos objetos. Normalmente são utilizados na validação de um diagrama de classes ou identificação de problemas na execução de uma aplicação.

O diagrama de estado é responsável por descrever o comportamento de objetos em relação a eventos. Ele apresenta o ciclo de vida de um objeto mostrando os eventos que causam a transição de um estado para outro estado e as ações que resultam de uma mudança de estado. Este diagrama não é escrito para todas as classes de um sistema e sim somente para aquelas que possuem um número definido de estados conhecidos e onde o comportamento das classes é afetado pelos diferentes estados.

O diagrama de sequência é responsável por apresentar a colaboração dinâmica entre os objetos de um sistema. Sua principal característica é a possibilidade de visualização da sequência de mensagens trocadas entre os objetos.

O diagrama de colaboração é semelhante ao diagrama de sequência. No diagrama de colaboração também estão inclusos os relacionamentos entre os objetos. Normalmente pode-se escolher entre utilizar o diagrama de colaboração ou o diagrama de sequência.

O diagrama de atividades permite descrever as operações associadas a um objeto e também suas interações com outros objetos. Este diagrama é uma variação do diagrama

de estado com o objetivo de capturar ações e seus resultados em termos de mudanças de estados dos objetos.

O diagrama de componente e de execução tem como objetivo mostrar o lado funcional do sistema, descrevendo a relação entre seus componentes e organização de seus módulos durante sua execução. O diagrama de componentes mostra a organização estrutural de todo o código gerado para aplicação. Já o diagrama de execução apresenta a arquitetura física do *hardware* e do *software* do sistema, incluindo diversos periféricos, computadores e as conexões estabelecidas entre eles.

Como já apresentado, a importância da informática na gestão das empresas é inquestionável. As empresas ganham agilidade, confiabilidade e ainda podem reduzir seus custos se o gerenciamento for feito de forma eficaz. Para se obter um gerenciamento eficaz, é necessário ter um *software* que forneça dados consistentes e de forma organizada. A (SBSISTEMAS, 2015) destaca como a utilização de um *software* de gestão pode auxiliar na tomada de decisões, organização de planejamento estratégico e ainda descobrir quais áreas da empresa precisam de maior atenção. Além de todas essas vantagens, um *software* de gestão oferece a possibilidade de gerenciamento de permissões de acesso, assim é possível controlar o que cada funcionário tem acesso. O sistema a ser desenvolvido neste projeto, contará com esse gerenciamento de permissões de acesso. Os funcionários não terão acesso as funções dos administradores.

3.11 Banco de dados e *driver JDBC*

Em muitos sistemas computacionais o banco de dados pode ser considerado como a parte mais importante da aplicação, o sucesso do sistema está totalmente relacionado com o bom funcionamento do banco de dados, conforme descreve (OLIVEIRA, 2004). No projeto em questão o banco de dados terá papel fundamental no armazenamento de tabelas, relatórios e/ou dados que possam auxiliar no processo de gerenciamento da empresa.

Um banco de dados pode ser definido como uma coleção de dados organizados de forma sistemática. Esses dados são acessados e gerenciados por um **Sistema de Gerenciamento de Banco de Dados (SGBD)**. O sistema de gerenciamento provê mecanismos de organização, recuperação, alteração e armazenagem de dados. As consultas em bancos de dados relacionais são realizadas em uma linguagem de padrão internacional conhecida como SQL (COSTA, 2011). O processo de funcionamento de um SGBD é apresentado de forma sucinta por (OLIVEIRA, 2004, p. 2):

”Aplicações que utilizam SGBDs para gerenciar dados executam processos separados para conectar com o SGBD. Uma vez que uma conexão é estabelecida, comandos SQL podem ser usados para inserir, apagar ou

modificar dados. Consultas **SQL** podem ser usadas para restaurar dados desejados, mas, mas é necessário exibir uma importante diferença de como o sistema de banco de dados “vê” os dados e como uma aplicação desenvolvida em linguagens como **C++** ou **Java** vêem os dados: o resultado de uma consulta no banco de dados é um conjunto (ou coleção) de registros, mas linguagens estruturadas com **Java**, por exemplo, não possui um tipo de dado correspondente. Esta “falta de combinação”, conhecida como *impedance mismatch*, entre as linguagens é resolvida através de construções adicionais **SQL** que possibilitam às aplicações reconhecer e acessar um conjunto e interagir com um registro por vez.“

Alguns dos sistemas de gerenciamento de banco de dados mais utilizados no mundo são: **Microsoft SQL Server**, **Oracle**, **MySQL** e **PostgreSQL**. Neste trabalho o sistema de gerenciamento a ser utilizado será o **PostgreSQL**.

O livro (DEITEL; DEITEL, 2010, p. 900) descreve um banco de dados relacional como uma representação lógica de dados, que permite que os dados sejam acessados sem considerar sua estrutura física. Os dados em um banco de dados são armazenados no formato de tabelas.

Existem duas principais formas de se armazenar dados atualmente, a primeira delas é fazendo o uso de arquivos de dados permanentes e a segunda é a utilização de um banco de dados. O banco de dados apresenta inúmeras vantagens em relação a um sistema de arquivos como: a centralização dos dados, eliminação de inconsistências, controle de redundância, independência de dados e facilidade de acesso (COSTA, 2011).

A centralização dos dados ocorre pelo fato de todos os dados estarem concentrados em um único local, já no sistema de arquivos cada aplicação mantém seus próprios arquivos.

A garantia da integridade em um banco de dados está diretamente relacionada a consistência dos dados. Em um sistema de arquivos pode ocorrer a repetição da informação armazenada fazendo com que um mesmo dado apresente valores diferentes dependendo de como esses arquivos são acessados. Em um banco de dados o mesmo não ocorre garantindo a integridade das informações e a consistência dos dados.

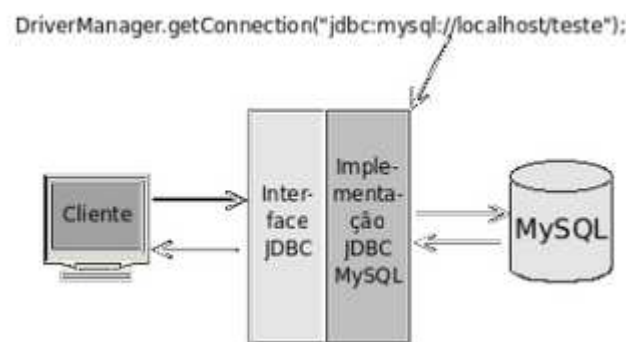
Em um sistema de arquivos geralmente uma mesma informação pode estar presente em um ou mais arquivos, causando um desperdício de espaço de armazenamento. No banco de dados, o dado é armazenado apenas uma vez e pode ser compartilhado por diversos usuários evitando assim a redundância.

A definição da estrutura de armazenamento e do método de acesso aos dados em um sistema de arquivos, está inclusa no código da aplicação. Para fazer quaisquer alterações nas definições ou métodos de acesso, é necessário modificar a aplicação, gerando mais trabalho e consumindo mais tempo. No banco de dados o mesmo não ocorre, pois, os dados são independentes da aplicação. Este fato recebe o nome de abstração de dados.

O processo de armazenamento dos dados também é conhecido como persistência. Em Java a biblioteca padrão de persistência em um banco de dados relacional recebe o nome de JDBC (Java Database Connectivity). Com o objetivo de evitar que cada banco de dados tenha sua API, conjunto de classes e métodos, existe um único conjunto de interfaces muito bem definidas. Este conjunto de interfaces é representado como um *driver* (JDBC). Normalmente os sistemas de gerenciamento de banco de dados fornecem seus *drivers* (CAELUM, b).

Entre as inúmeras interfaces deste pacote pode-se destacar a "Connection". Esta e outras interfaces são responsáveis por fazer a ponte entre o código do cliente que usa a API JDBC e o banco de dados, como mostra a Figura 24.

Figura 24 – Interação entre o banco de dados e o cliente.



Fonte: (CAELUM, b).

3.12 A linguagem CSS

Quando a linguagem HTML surgiu, seu objetivo era apenas de mostrar um determinado conteúdo para um usuário, sem se preocupar com a formatação e apresentação dos dados na tela. A medida em que o tempo passou, surgiram novas necessidades de formatação e apresentação de dados relacionadas a linguagem HTML. A própria linguagem HTML passou a ser utilizada para o controle da aparência do documento tornando a linguagem mais complexa conforme descreve (BARROS, 2008).

No ano de 1994, um desenvolvedor chamado de Hakon Lie propôs a criação da linguagem CSS (Cascade Style Sheets). A CSS é uma linguagem de estilos utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML e XML. Com o surgimento da CSS, os códigos HTML passaram a ter *links* (ligação) para um arquivo CSS que contém os estilos de uma respectiva página HTML. Com isso foi possível separar o formato de uma página HTML de seu conteúdo, tornando assim o código mais organizado, funcional e simples (BARROS, 2008).

A utilização da linguagem CSS na definição de estilos de páginas HTML apresenta algumas vantagens como:

- Controle de *layout* de vários documentos partir de um único arquivo CSS;
- Facilidade de implementação por ser estruturado em classes;
- Rapidez por ter a possibilidade de utilizar apenas uma classe para muitos documentos, diminuindo assim conteúdo desnecessário;
- Boa portabilidade por funcionar em qualquer navegador;
- Precisão no controle do *layout*.

Neste projeto a linguagem CSS terá papel fundamental na criação da interface, objetivando proporcionar ao usuário uma experiência simples e agradável de navegação no sistema *Web*.

3.13 Ferramentas de Desenvolvimento

Para o desenvolvimento de um sistema *Web* completo, incluindo banco de dados, servidor *Web*, periféricos e aplicação *Java Web*, é necessário a utilização de algumas ferramentas. Estas ferramentas são *softwares* que tem como objetivo tornar o desenvolvimento rápido, seguro e eficiente.

Entre tais ferramentas estão inclusas ambientes integrados de desenvolvimento, *plug-ins* e alguns *softwares*. Nas próximas seções serão apresentadas as ferramentas utilizadas no desenvolvimento deste projeto.

3.13.1 Ferramenta de desenvolvimento - *Eclipse IDE*

O Eclipse é uma IDE (Integrated Development Environment) baseada em Java líder de mercado. Formada por um consórcio de empresas lideradas pela IBM possui seu código livre. Esta IDE compreende os mais diversos tipos de linguagens como Java, C/C++, PHP, Python, Pearl e ainda aceita a instalação de *plug-ins* que auxiliam no desenvolvimento de aplicações.

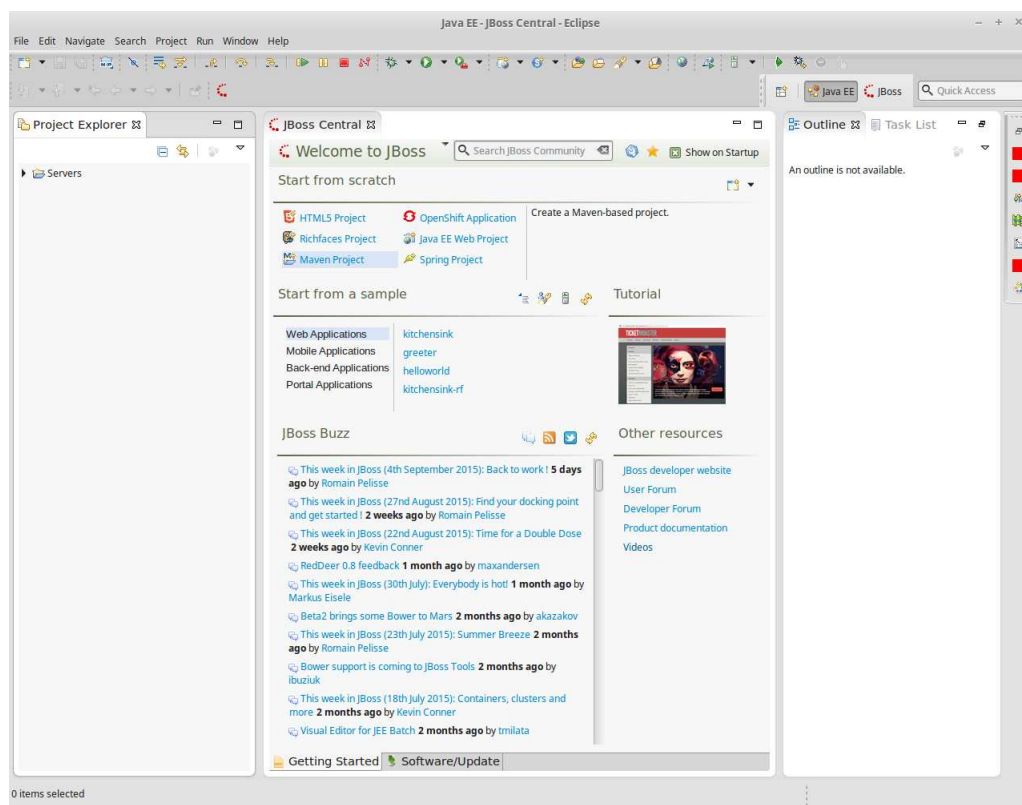
O principal diferencial do Eclipse é apresentado segundo (CAMPOS, 2003):

” O grande diferencial do Eclipse é a sua neutralidade em relação à linguagem de programação e ao sistema operacional. O Eclipse Consortium, formado pela IBM, Merant, HP, Borland, Fujitsu, Red Hat, Sybase, Oracle, Rational e outras dezenas de empresas e organizações, fornece suporte a Java, C e COBOL nas plataformas Windows, Linux, AIX, Solaris, HP-UX, QNX e MacOS. Mas, como o *framework* é livre, qualquer

um pode adicionar o suporte a seu ambiente preferido, e já existem mais de 260 *plug-ins* para o Eclipse disponíveis no SourceForge e outros sites para desenvolvedores Perl, Python, Pascal, PHP e até Clipper.“

Pelo fato do Eclipse ter sido escrito em Java, é necessário ter uma máquina virtual Java (JRE) para rodá-lo, por este fato surge sua independência do sistema operacional. Para facilitar o desenvolvimento *Web* será utilizado uma plataforma chamada Web Tools Platform (WTP). Esta plataforma é um conjunto de *plug-ins* que têm como objetivo tornar o desenvolvimento de aplicações Java EE mais atrativas. Neste pacote estão inclusos editores para as linguagens que serão utilizadas neste projeto como JSP, CSS, HTML e Java. Existe uma versão do Eclipse (Eclipse IDE for Java EE Developers) que já vêm com esse *plug-in* em sua versão padrão. A Figura 25 apresenta a interface do Eclipse IDE for Java EE Developers.

Figura 25 – Interface da Eclipse IDE.



Fonte: Autoria própria.

3.13.2 Servidor Web - Apache Tomcat

O Apache Tomcat é um servidor Java *Web* desenvolvido pela Apache Software Foundation. É um *software* livre que têm como habilidade converter páginas JSP em um Servlet. Pode-se dizer que ele gera códigos em Java a partir de códigos em HTML como apresenta (DEVMEDIA, 2014).

O autor (COUTINHO, 2002a) define o Tomcat como sendo “é um *Servlet Container*, ou seja, é um servidor onde são instaladas *Servlets* para tratar as requisições que o servidor receber”.

A dinâmica do funcionamento de um processo de acesso à páginas *Web* segue a seguinte lógica: primeiramente o usuário faz uma solicitação de uma página indicada por meio de uma URL no *browser*; o servidor **Apache Tomcat** recebe a solicitação do usuário, executa o *Servlet* ou *JSP* associado a respectiva URL e então, o conteúdo gerado pelo *Servlet* ou *JSP* é apresentado no formato HTML; por fim, o usuário recebe o conteúdo gerado pelo Tomcat em seu *browser* (DEVMEDIA, 2014).

3.13.3 Sistema de Gerenciamento de banco de dados - *PostgreSQL*

O sistema gerenciador de banco de dados **PostgreSQL** surgiu na **Universidade de Berkeley**, na Califórnia, em 1986. Um desenvolvedor de nome **Michael Stonebraker** foi responsável por liderar uma equipe de desenvolvimento para criação de um banco de dados relacional. Surgiu então a primeira versão de um banco de dados relacional, chamada de **Postgres**. Posteriormente dois estudantes de **Berkeley** incorporaram a linguagem SQL ao **Postgres**, dando origem ao **Postgres95**. Em 1996 uma nova versão foi lançada definindo a linguagem SQL como padrão, esta versão recebeu o nome de **PostgreSQL** conforme explica (ALECRIM, 2006).

O **PostgreSQL** é gratuito e possui seu código aberto. Atualmente é um banco de dados que suporta a maior quantidade de arquiteturas e *software* do mercado, além de possuir uma comunidade de suporte a assuntos relacionados ao **PostgreSQL** (COUTINHO, 2002b). Normalmente é utilizado em aplicações complexas com grande volume de dados como discute (ALECRIM, 2006).

3.13.4 Ferramenta de gerenciamento do *PostgreSQL* - *pgAdmin*

Para realizar a administração do banco de dados de uma maneira mais eficiente, será utilizada uma ferramenta **pgAdmin**. Esta ferramenta proporciona um ambiente visual gráfico de administração do sistema gerenciador de banco de dados **PostgreSQL**. Foi desenvolvida pela própria equipe de desenvolvimento do **PostgreSQL**.

4 Metodologia

O primeiro passo no desenvolvimento do projeto será a realização de um estudo a fim de se definir quais sensores serão utilizados. Serão realizados alguns testes com sensores de proximidade indutivos, capacitivos, ultrassônicos e de luz infravermelha para verificar se estão de acordo as necessidades e realidade do cliente. Também serão realizados testes preliminares, relacionados aos aspectos de comportamento da placa **CORTEX M3** perante a algumas situações relacionadas a questões de tempo de resposta e erros de aquisição de dados. Todos os ensaios serão realizados no laboratório de eletrônica da universidade.

Concluída a fase de testes dos sensores do sistema de *hardware*, o próximo passo consistirá no desenvolvimento do sistema supervisor, que será descarregado na placa de aquisição de dados. A plataforma utilizada para o desenvolvimento do *software* será a IDE **Atmel Studio**, que é projetada pela Somnium exclusivamente para controladores ARM desse fabricante. A implementação do código será realizada utilizando a linguagem de programação **C**. O objetivo do *software* consiste em obter a data e hora em que os eventos (entrada e saída de clientes) captados pelos sensores ocorrem. Nesta etapa também será analisada a necessidade do uso de interrupções para obtenção dos dados de forma segura e eficiente.

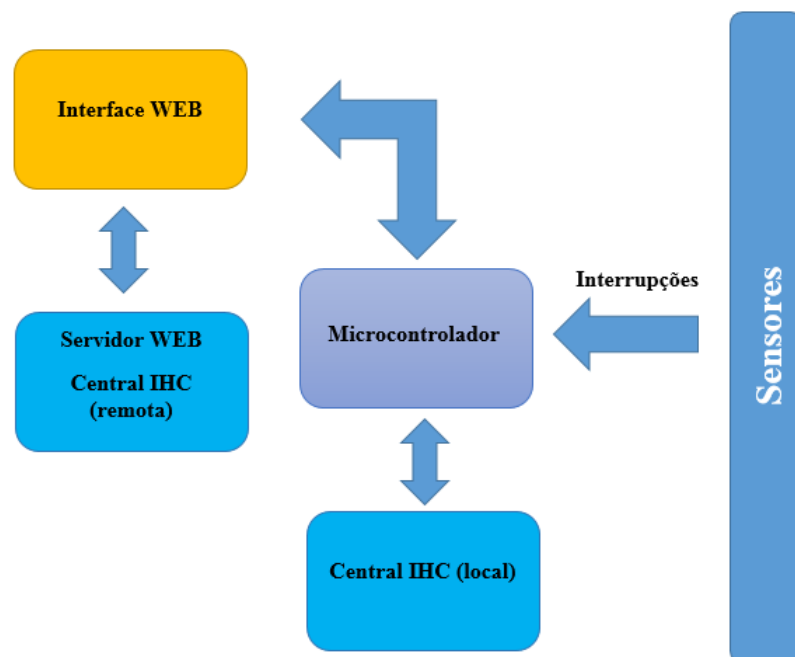
Com o microcontrolador programado e em perfeita sincronia com os sensores, será iniciado o desenvolvimento da segunda parte do projeto que consiste no *software* de gerenciamento e manipulação de dados. Tal *software* será implementado em uma interface **Web** e terá o seu desenvolvimento norteado de acordo com os conceitos de engenharia de *software*. A interface terá como principal objetivo proporcionar uma visão macro do controle de fluxo de clientes em tempo real. Nessa interface serão priorizadas ferramentas que auxiliem a geração de relatórios e que descrevam o comportamento do sistema, permitindo a estipulação de novas estratégias para aumento de eficiência e lucro.

A etapa seguinte será a realização de ensaios envolvendo aquisição, armazenamento, processamento e interpretação dos dados adquiridos. Obtendo êxito nesse ponto, o passo final consistirá na implementação real do sistema na rede de motéis.

5 Desenvolvimento

A Figura 26 apresenta a topologia básica do projeto abordado neste trabalho por meio de um diagrama de blocos. O conjunto é composto por um microcontrolador **Arduino DUE** responsável pela aquisição dos eventos gerados pelos sensores. Na seção 27 são apresentados mais detalhes a respeito desse microcontrolador. O bloco de sensores é constituído em primeiro momento por um conjunto de seis sensores magnéticos que serão instalados nos portões. Esse tipo de sensor é bastante utilizado nesse tipo de aplicação (detecção de abertura/fechamento de portões, portas e janelas). Seu funcionamento se dá de maneira bem simples, a partir de um campo magnético geralmente proveniente de um ímã permanente ou de uma bobina. Tem o mesmo princípio de funcionamento de um gerador elétrico. A interface **Web** é composta por um módulo **ethernet R3**, seu **chip Wiznet W5100** fornece uma rede IP compatível com os protocolos TCP e UDP. Conforme apresenta a sua documentação, esse **chip** tem como característica a sua facilidade de integração, programação e grande estabilidade. A central **IHC** local será composta por um display **LCD** que exibirá algumas informações úteis em tempo real relacionadas ao status do sistema. Os eventos detectados e tratados pelo microcontrolador compõem a base de dados do servidor. A partir desses dados uma interface **IHC** remota os apresentará de maneira amigável ao operador do sistema.

Figura 26 – Topologia do projeto.



Fonte: Autoria própria.

5.1 Arduino DUE

O microcontrolador escolhido para compor a central de aquisição de dados é o Arduino DUE, equipado com um chip de 32 *bits* ARM Cortex-M3, mais precisamente o modelo Atmel SAM3X8E fabricado pela própria Atmel. A escolha dessa plataforma pode ser justificada devido ao seu baixo custo de aquisição, alto poder de processamento e facilidade de configuração. Foi o microcontrolador que apresentou a melhor relação custo x benefício dentre os analisados (modelos da família PIC e alguns modelos fabricados pela Freescale). O seu núcleo permite que operações de 4 *bytes* sejam realizadas em um único clique de *clock*. A frequência de operação nativa é de 84 *MHz*. Entre as suas demais especificações, estão uma memória *flash* de 512 *KB* e 96 *KB* de *SRAM* que garante muito mais recursos para programas maiores e mais complexos. A Figura 27 apresenta uma visão da frente e do verso da placa.

Figura 27 – Arduino DUE.



Fonte: (LIMA, 2014).

A placa possui um total de 54 pinos digitais de I/O, em que desses, 12 podem ser utilizados como controle PWM. O seu conversor ADC é incrivelmente rápido podendo operar em uma frequência de amostragem de até 1 *MHz* com 12 *bits* de resolução. Além disso, possui dois conversores DAC também de 12 *bits* de resolução. Possui 54 pinos digitais de entrada e saída. A tensão de operação dos pinos é de 3,3 V. Porém, uma das características que mais despertou atenção nesta placa é a possibilidade de configurar praticamente todos

os seus pinos para trabalharem com interrupções de *hardware*. Devido a sua robustez, o **Arduino DUE** é bastante utilizado em tarefas de captura e processamento de áudio ou no controle de vários motores.

6 Recursos

Na Tabla 1 pode-se visualizar a lista de recursos necessários para o desenvolvimento deste projeto de pesquisa.

Tabela 1 – Recursos necessários para o projeto.

Item	Quantidade	Valor unitário	Valor total
Módulo Ethernet	1	R\$ 54.60	R\$ 54.60
Display LCD	1	R\$ 20.00	R\$ 20.00
Protoboard	1	R\$ 75.10	R\$ 75.10
Hospedagem Web Site	1	R\$ 214.80 (anual)	R\$ 214.80
Sensores Magnéticos	6	R\$ 6.12	R\$ 36.74
Arduino Due	1	R\$ 135.00	R\$ 135.00
Fios	-	-	R\$ 10.00
Parafusos	-	-	R\$ 27.00

Fonte: Autor deste estudo.

Os recursos apresentados na Tabla 1 serão adquiridos pelo o autor desse projeto.

7 Cronograma

Na Tabla 2 é apresentado o cronograma de desenvolvimento deste projeto de pesquisa.

Tabela 2 – Cronograma do projeto.

Cronograma - Trabalho Final de Graduação										
Atividade	2015 (meses)									
	3	4	5	6	7	8	9	10	11	
Projeto de Pesquisa	X									
Estudo de conceitos a serem utilizados	X	X	X	X	X	X	X	X		
Monografia Parcial				X	X	X				
Desenvolver protótipo do sistema						X	X			
Programação do microcontrolador para obtenção de dados do sensor						X	X			
Desenvolvimento dos diagramas de caso de uso pra interface Web						X	X			
Criação e implementação do banco de dados so sistema						X	X			
Desenvolvimento de telas da interface Web						X	X	X		
Desenvolvimento das interfaces projetadas						X	X	X		
Testes Finais e Implementação								X	X	
Defesa da monografia Final										X

Fonte: Autoria própria.

8 Resultados Esperados

Pretende-se obter um sistema para controle de fluxo de clientes eficiente, de baixo custo, seguro e confiável. A construção de um servidor com interface **Web** auxiliará no gerenciamento em tempo real do sistema, trazendo agilidade, rapidez e facilidade no monitoramento de eventos.

Após a conclusão do projeto, espera-se obter um produto final que possa ser vendido para empresas que apresentem problemas de gerenciamento semelhantes aos discutidos neste trabalho.

O desenvolvimento desse sistema viabilizará maior integração entre os conceitos estudados no curso de engenharia da computação, como engenharia de software, banco de dados, linguagem **Java** e **C**, sistemas embarcados, redes, arquitetura de computadores, gestão de projetos e microcontroladores. Assim pretende-se solidificar o conhecimento adquirido ao decorrer de todo o curso.

Referências

- ALECRIM, E. Banco de dados mysql e postgresql. *Info Wester*, Novembro 2006. Citado na página 49.
- BARALE, R. F. Desenvolvimento de um sistema de vendas na web utilizando jsp. 2007. Citado na página 37.
- BARRETTO, M. R. P.; MIYAGI, P. E.; SILVA, J. R. Domótica - controle e automação. In: PABI, E. do R. T. (Ed.). [S.l.: s.n.], 1993. v. 1, p. 274. Citado na página 13.
- BARROS, C. F. A. d. S. Isabelle Guimarães M. O. de. *Apostila de Introdução ao CSS*. Rio de Janeiro, 2008. Citado na página 46.
- BEITOL, W. Os efeitos da automação do ponto-de-vista nas vendas e no lucro. faith system. 2015. Disponível em: <http://www.faithsystem.com.br/artigos_sobre_forca_de_vendas_e_mobilidade_os_efeitos_da_automacao_do_ponto_de_vista_nas_vendas_e_no_lucro.asp>. Acesso em: 10 de Abril. 2015. Citado na página 9.
- CAELUM. *Java e Orientação a Objetos*. Disponível em: <<http://www.caelum.com.br/apostila-java-orientacao-objetos/>>. Acesso em: 20 de Agosto. 2015. Citado na página 34.
- CAELUM. *Java Web*. Disponível em: <<http://www.caelum.com.br/apostila-java-web/>>. Acesso em: 22 de Agosto. 2015. Citado 3 vezes nas páginas 37, 45 e 46.
- CAMPOS, A. Eclipse o super ide. *Embase: VI EBAI*, Maio 2003. Citado na página 47.
- COLE, A. Modelo cascata. Março 2011. Disponível em: <<https://anielacole.wordpress.com/2011/03/25/modelo-cascata-parte-ii>>. Acesso em: 25 de Abril. 2015. Citado na página 41.
- COSTA, E. R. D. Bancos de dados relacionais. In: . São Paulo: [s.n.], 2011. Disponível em: <<http://www.fatecsp.br/dti/tcc/tcc0025>>. Citado 2 vezes nas páginas 44 e 45.
- COUTINHO, F. Introdução ao tomcat e servlets. *Embase: VI EBAI*, Agosto 2002. Citado na página 48.
- COUTINHO, F. Introdução ao tomcat e servlets. *Embase: VI EBAI*, Agosto 2002. Citado na página 49.
- DEITEL, P.; DEITEL, H. *Java Como Programar*. 8. ed. [S.l.]: Pearson Prentice Hall, 2010. ISBN 9788576051947. Citado na página 45.
- DEVMEDIA, E. Conheça o apache tomcat. DevMedia, 2014. Disponível em: <<http://www.devmedia.com.br/conheca-o-apache-tomcat/4546>>. Citado na página 48.
- ELETRÔNICA, S. *Novo microcontrolador MSP430 da Texas facilita conectividade*. 2011. Disponível em: <<http://www.sabereletronica.com.br/noticias-5/416-novo-microcontrolador-msp430-da-texas-facilita-conectividade>>. Citado na página 19.

- FALBO, R. d. A. *Engenharia de software: notas de aula*. 2005. Disponível em: <<http://www.inf.ufes.br/~falbo/download/aulas/es-g/2005-1/NotasDeAula.pdf>>. Acesso em: 25 de Abril. 2015. Citado na página 41.
- JEVEAUX, P. C. M. *Projeto: Tecnologia Empresarial Material de Estudo: Java Server Pages (JSP)*. 2008. Citado 2 vezes nas páginas 38 e 39.
- KUROSE, J.; ROSS, K. *Redes de computadores e a internet: uma abordagem top-down*. Pearson, 2010. ISBN 9788588639973. Disponível em: <<https://books.google.com.br/books?id=raZtQwAACAAJ>>. Citado 3 vezes nas páginas 31, 32 e 33.
- LIMA, C. B. de; VILLAÇA, M. V. M. *AVR e Arduino*. 1. ed. [S.l.]: LTC, 2012. Citado 2 vezes nas páginas 20 e 21.
- LIMA, T. *ARM Cortex M0+*. 2013. Disponível em: <<http://www.embarcados.com.br/arm-cortex-m0/>>. Acesso em: 16 de Abril. 2015. Citado na página 19.
- LIMA, T. *Arduino Due*. 2014. Disponível em: <<http://www.embarcados.com.br/arduino-due/>>. Acesso em: 4 de Setembro. 2015. Citado na página 54.
- MACHADO, F. B.; MAIA, L. P. *Arquitetura de Sistemas Operacionais*. 4. ed. [S.l.]: LTC, 2007. Citado 9 vezes nas páginas 20, 21, 23, 24, 25, 27, 28, 29 e 30.
- MARIMOTO, C. E. Processadores risc x processadores cisc. Julho 2007. Disponível em: <<http://www.hardware.com.br/artigos/risc-cisc/>>. Acesso em: 25 de Abril. 2015. Citado na página 15.
- MARTINS, M. *MVC simples e prático*. 2011. Disponível em: <<http://www.k19.com.br/artigos/mvc-simples-e-pratico-parte-i/>>. Citado na página 40.
- MEDEIROS, H. Introdução ao padrão mvc. DevMedia, 2014. Disponível em: <<http://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>>. Citado na página 40.
- OLIVEIRA, A. de; ANDRADE, F. de. *Sistemas embarcados: hardware e firmware na prática*. 2. ed. Editora Érica Ltda, 2014. ISBN 9788536501055. Disponível em: <<https://books.google.com.br/books?id=u9KVGAAACAAJ>>. Citado 2 vezes nas páginas 14 e 16.
- OLIVEIRA, M. da S. Desenvolvimento de aplicações de banco de dados. *Instituto de Computação - Universidade Estadual de Campinas*, São Paulo, 2004. Disponível em: <<http://www.ic.unicamp.br/~geovane/mo410-091/Ch06-DBApp-art.pdf>>. Acesso em: 24 de Abril. 2015. Citado 2 vezes nas páginas 35 e 44.
- ORACLE. *ClassHttpServlet*. [S.l.], 2015. Disponível em: <<http://docs.oracle.com/javase/7/api/javax/servlet/http/HttpServlet.html>>. Acesso em: 4 de Setembro. 2015. Citado na página 38.
- RESENDE, A. M. P. de; SILVA, C. C. da. *Programação orientada a aspectos em Java*. [S.l.]: Brasport, 2005. Citado na página 36.
- SBSISTEMAS. Como um software de gestão pode fazer a diferença no seu controle financeiro. Março 2015. Disponível em: <<http://www.sbsistemas.com.br/blog/software-de-gestao-fazer-diferenca-controle-financeiro/>>. Acesso em: 25 de Abril. 2015. Citado na página 44.

SOARES, S. et al. Implementando sistemas orientados a objetos para web usando servlets e java. *Centro de Estudo e Sistemas Avançados do Recife*. Citado na página 35.

STALLINGS, W. *Arquitetura e Organização de Computadores*. 5. ed. [S.l.]: Pearson Prentice Hall, 2003. Citado 2 vezes nas páginas 22 e 29.

UML: Linguagem de Modelagem Unificada. Disponível em: <http://www.etelg.com.br/paginaete/downloads/informatica/apostila_uml.pdf>. Citado na página 42.

VASILJEVIC, G. *Apostila de Arduino*. 2013. Disponível em: <http://nilsinho.com/Arduino/Apostilas/apostila_v0.5a.pdf>. Acesso em: 2 de Abril. 2015. Citado na página 17.

WHEAT, D. *Arduino internals*. [S.l.]: Apress, 2011. Citado na página 18.