Índice de Figuras

Índice de Tabelas

1 Introdução

1.1 Ensino a distância

Ensino à distância é uma realidade emergente na qual estudantes, professores e pesquisadores, em diferentes localidades, podem interagir por meio do acesso remoto de recursos dedicados ao ensino. Recentemente, devido ao desenvolvimento na infra-estrutura de comunicações, as funções oferecidas pelo ensino à distância abrangem inclusive a interação com laboratórios reais, nos quais processos físicos podem ser controlados remotamente.

Ao longo da vida profissional, a renovação constante de conhecimento tem se tornando um paradigma. Métodos convencionais de ensino são baseados em palestras, conferências e publicações estáticas (eg. livros). A aplicação exclusiva desta metodologia tradicional, especialmente em nível superior, é em geral lenta, cara e conseqüentemente ineficiente diante do universo de informações necessários ao atual contexto pedagógico. Novas metodologias e mídias não convencionais ocupam cada vez mais espaço no atual processo de ensino, especialmente nas universidades. São exemplos destas ferramentas modernas: livros eletrônicos, teleconferências, sistemas multimídias, simuladores, laboratórios remotos, *softwares* educacionais, etc. Já podemos observar inclusive, universidades oferecerem cursos completos de nível superior ministrados exclusivamente por meio do ensino à distância.

Experimentos em laboratórios são fundamentais no processo de educação de engenheiros, uma vez que os estudantes podem aplicar e verificar na prática a teoria adquirida.

O público alvo deste projeto são estudantes cursando disciplinas envolvidas na área de automação e controle. Estes estudantes realizarão experimentos em casa, no campus universitário ou em qualquer lugar do mundo com apenas um computador na *internet*. Os alunos farão ajustes *on-line* nos parâmetros de controle e obterão a resposta completa (gráfica, matemática e dados) do experimento em tempo real por meio do WEB *Browser*.

A simulação é tão boa quanto for o seu modelo matemático, entretanto a experimentação tem a vantagem de oferecer ao usuário situações e fenômenos que são difíceis ou impossíveis de simular. A experimentação remota permite que o uso de equipamentos caros ou únicos seja compartilhado entre diferentes instituições (eg. universidades). A filosofia do uso remoto de laboratórios está intimamente ligada ao controle de sistemas físicos e suas aplicações extrapolam o universo do ensino à distância. Muitas são as aplicações em outras áreas, como industrial, médica, comercial e entretenimento.

A idéia principal de um laboratório remoto é usar a *World Wide* WEB como plataforma de comunicação e o WEB *Browser* como sua interface. A *internet* provê a plataforma para transmissão de informações enquanto o WEB *Browser* é o próprio ambiente para execução do *software* cliente. O servidor WEB é o intermediário entre o cliente e o experimento. Nesse sentido, serão estudadas as diversas soluções computacionais razoáveis com ênfase na interação em tempo real entre o usuário e o processo. Mais ainda, premissas educacionais e computacionais mínimas associadas à qualidade de ensino a distância serão investigadas e adotadas neste projeto.

1.2 Inteligência Artificial

Lógica *Fuzzy*, Redes Neurais, Sistemas Especialistas e Algoritmos Genéticos fazem parte de um novo paradigma conhecido por *sistemas inteligentes* [20]. Este conjunto de paradigmas tem como objetivo justificar como um comportamento inteligente pode emergir de implementações artificiais, fornecendo respostas que solucionam problemas, de forma apropriada às situações específicas destes problemas, mesmo sendo novas ou inesperadas.

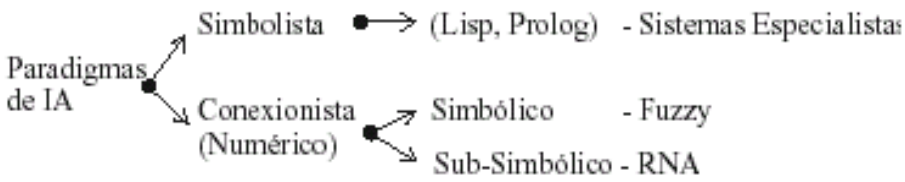

Figura 1 : Paradigmas de Inteligência Artificial

A figura 01 mostra os dois campos de paradigmas da inteligência artificial. A abordagem Conexionista considera ser virtualmente impossível transformar em algoritmo, isto é, reduzir a uma seqüência de passos lógicos e aritméticos diversas tarefas que a mente humana executa com facilidade e rapidez - como por exemplo: reconhecer rostos, evocação de memória pela associação etc.- enquanto que a abordagem Simbolista acredita que através de métodos heurísticos pode-se solucionar tais problemas.

Os sistemas inteligentes têm como características principais a capacidade de aprender, de se adaptar a um ambiente desconhecido ou a uma nova situação. Não é tão simples classificar sistemas em sistemas inteligentes, pois a maioria das características de um sistema inteligente estão presentes nos sistemas clássicos (controles PD, PI, PID). É mais fácil, e mais lógico, classificar os sistemas que não são inteligentes.

Uma subdivisão da abordagem Conexionista está representada na figura 01, na qual tem-se classificadas Redes Neurais Artificiais (RNA) e Lógica *Fuzzy*. As Redes Neurais Artificiais aprendem a heurística a partir dos dados, enquanto que a Lógica *Fuzzy* gera a representação de conhecimentos heurísticos por meio de regras.

A operação destes sistemas é inspirada, em geral, por sistemas biológicos. A capacidade criativa dos seres humanos, de raciocinar de maneira imprecisa, incerta ou difusa contrasta com a forma de operar de computadores e máquinas, movidos por lógica binária e precisa. No momento em que estas máquinas perdessem esta restrição, tornar-se-iam eventualmente inteligentes, podendo raciocinar de forma imprecisa. Esta forma de raciocínio é conhecida em inglês por *Fuzzy*, tendo como tradução mais próxima a palavra "nebuloso".

A Lógica *Fuzzy*, então, procura incorporar a forma humana de pensar em sistemas. Quando esta técnica é aplicada em malhas de controle, é chamada de Controle *Fuzzy*. Os controladores produzidos geralmente têm uso em aplicações constituídas por sistemas dinâmicos complexos. Uma aplicação de sistemas *Fuzzy* bem conhecida é o *Controle Fuzzy Supervisório*, ou Operacional, onde plantas industriais complexas são automatizadas em funções delegadas tradicionalmente a operadores humanos. O controlador *Fuzzy* busca, então, capturar a experiência do operador humano fornecendo uma técnica para projeto de algoritmos de supervisão.

Uma técnica de controle inteligente também bastante conhecida consiste nas Redes Neurais Artificiais as quais são utilizadas em aplicações como o reconhecimento de padrões (caractere, voz, faces), controle/robótica, processamento de sinais, otimizações e associação de padrões, simulando funções biológicas do cérebro humano para execução das tarefas de controle. Esta técnica procura aprender como controlar um sistema através de exemplos numéricos, enquanto que lógica *Fuzzy* aprende através de exemplos semânticos.

A associação entre estas duas técnicas recebe o nome de sistema "*Neurofuzzy*" que combina as vantagens dos dois tipos de sistemas inteligentes. Neste sistema a lógica *fuzzy* providencia os fundamentos teóricos para a captura de incertezas associadas com os processos de pensamento humano, através do emprego de definições lingüísticas de variáveis utilizadas em um sistema com uma base de regras. Já as redes neurais treinadas com os conjuntos de dados que contém o comportamento desejado do sistema, podem extrair as regras da lógica *fuzzy* e passá-las ao controlador *fuzzy*. Teoricamente, a combinação dessas duas tecnologias pode conter uma resposta completa aos problemas de projeto de sistemas inteligentes.

1.3 Controle Clássico e Controle Inteligente

Atualmente, diversas técnicas de controle, embasadas na teoria de controle clássico estão disponíveis. Estas técnicas usam as características do sistema linear a ser controlado como fundamento para projeto dos controladores lineares. O projeto dos controladores faz uso de diversas ferramentas, como LGR – lugar geométrico das raízes, e diagramas de Bode. Estas ferramentas fornecem justificativas para o emprego de uma dentre diversas técnicas de controle linear, como: "Compensação de Atraso e/ou Avanço", Controle PID, etc.

Especificações de desempenho e robustez para o processo em malha fechada são alcançadas partindo-se do desempenho do processo em malha aberta, ou seja, sem realimentação.

Na maioria dos casos, porém, os processos possuem característica não-linear bastante acentuada e, para projeto de controladores lineares, o processo é linearizado em torno de um ponto de operação. Então, somente para este ponto de operação, os parâmetros de desempenho são obtidos com a inserção do controlador linear. Isto decorre do fato que o processo linearizado varia seus parâmetros para pontos de operação distintos. Entretanto, vários processos industriais possuem não-linearidades suaves, o que não prejudica tanto o desempenho dos controladores lineares.

Para processos de não-linearidade muito acentuada e com exigências de desempenho críticas, algoritmos de controle inteligente tornam-se uma alternativa adequada. A capacidade destes algoritmos de variar suas estratégias de controle de acordo com o processo controlado consiste em grande vantagem. Processos simples, entretanto, com comportamento aproximadamente linear e exigências de desempenho pouco restritas, não encontram justificativa em usar técnicas de controle inteligente, pois o custo computacional envolvido na inserção de regras de controle mais complexas pode inviabilizar seu emprego.

Este debate sobre viabilidade e adequação da utilização de técnicas de controle inteligente é intenso, tanto no meio acadêmico como em ambiente industrial. Discussões sobre o ganho de desempenho oferecido pelo emprego de controle *Fuzzy* podem ser

observados em [13]. Aspectos sobre as gerações de algoritmos *Fuzzy* e suas restrições de desempenho são questionados e conclusões sobre a adequação do emprego desta técnica são feitas.

1.4 Modelagem de Processos em Sistemas de Controle

Modelagem simplifica esquematicamente situações reais para facilitar e reduzir o custo da execução de ensaios e experimentos sobre os sistemas reais. Existem, convencionalmente, três enfoques usados para modelagem de plantas e processos: experimental, matemático (analítico) e heurístico.

O método experimental equivale a obter pontos discretos de uma curva característica de entrada e saída para um determinado processo. Este método encontra restrições no que diz respeito à possibilidade de execução de experimentos nos sistemas reais.

O método matemático, por sua vez, procura obter um modelo idealizado do processo a ser controlado, na forma de equações diferenciais que descrevem a dinâmica do mesmo. Para simplificação dos modelos construídos, supõe-se que o processo é linear em um determinado ponto de operação. Esta propriedade de linearidade fornece técnicas conhecidas e muito eficazes para obtenção de soluções analíticas ideais. Outra propriedade suposta, a de que o sistema seja não-variante no tempo, também constitui uma restrição para este tipo de modelagem, ao supor, por exemplo, que componentes não se desgastam. Em decorrência destas restrições, existem dificuldades no desenvolvimento de uma descrição matemática realista do processo.

O método heurístico consiste na realização de tarefas de acordo com experiência prévia, pelo uso de regras práticas e conhecidas. Estas regras, que associam conclusões com condições, assemelham-se às tabelas criadas pelo método experimental.

1.5 Microcontroladores e Microprocessadores

Na década de 70, começaram a ser utilizados microprocessadores em computadores com a finalidade de obter uma maior eficiência no processamento de dados. Os microprocessadores da linha *Intel* foram um dos precursores. Baseado na arquitetura de um microprocessador e seus periféricos, foi criado um componente que comportasse todo um sistema que equivalesse a um microprocessador e seus periféricos fisicamente em uma unidade. Desta forma, nasceu o que chamamos de microcontrolador.

Um microcontrolador pode ser definido como um dispositivo eletrônico "pequeno", possuidor de diversas características de computadores pessoais (Ex: memória, processador), utilizado para controle de processos. Como possui memória, este componente pode ser programado para realizar um certo número de tarefas simplesmente gravando o programa no microcontrolador. Portanto, este componente executará o programa gravado cada vez que for alimentado.

A principal vantagem deste dispositivo é conter, em apenas um circuito integrado, diversos módulos úteis para o controle de processos, como: conversor AD, PWM, relógios, contadores, portas de E/S, comunicação serial, etc. Então, apesar de ter uma ULA bem menos poderosa que microprocessadores, os microcontroladores possuem diversos recursos em uma única pastilha de silício.

Com o avanço da tecnologia e a utilização da eletrônica digital por grande parte das empresas, o emprego de microcontroladores vêm sendo muito requisitado para um melhor desenvolvimento da produção, diminuindo os custos e trazendo benefícios para as empresas que utilizam esse sistema. Outro fator importante a ser salientado é que os microcontroladores podem não somente ser usados em empresas de médio/grande porte, como também ser utilizados em vários projetos de eletrônica, na substituição de vários componentes digitais (projetos melhores acabados – microcontroladores ocupam menor espaço), maior eficiência e praticidade (execução de componentes via *software*) e o principal em qualquer campo da engenharia, uma excelente relação custo/benefício [18] e [20].

2 Fundamentos Teóricos: Automação

Hoje se entende por automação qualquer sistema, apoiado em computadores, que substitua o trabalho humano e que vise à solução rápida e econômica para alcance dos complexos objetivos das indústrias e do setor de serviços. Os Pequenos computadores especializados, os Controladores Lógicos Programáveis (CLP), permitem tanto o controle lógico quanto o controle dinâmico, com a enorme vantagem de permitir ajustes mediante simples reprogramações, na própria instalação.

A automação implica a implantação de sistemas interligados e assistidos por redes de comunicação, compreendendo sistemas supervisórios e interfaces homem-máquina que possam auxiliar os operadores no exercício de supervisão e análise dos problemas que por ventura venham a ocorrer.

2.1 CLP

O CLP é um dispositivo digital que pode controlar máquinas e processos. Utiliza uma memória programável para armazenar instruções e executar funções específicas como controle de energização/desenergização, temporização, contagem, seqüenciamento, operações matemáticas e manipulação de dados.

Os CLPs permitem reduzir os custos dos materiais, de mão-de-obra, de instalação e de localização de falhas e reduzir as necessidades de fiação e os erros associados. Além dessas vantagens os CLPs ocupam menos espaço que os contadores, temporizadores, e outros componentes de controles utilizados anteriormente e possuem uma linguagem de

programação baseada em diagrama *Ladder* (lógica de relés) e símbolos elétricos consagrados.

Os CLP's também chamados de PLC's (*Programmable Logic Controler*) têm como partes básicas e características funcionais os módulos de I/O, CPU, Fonte, Memória, *Rack*. Funcionalmente um PLC opera da seguinte forma: a CPU (Unidade Central de Processamento) possui uma rotina interna que lê o estado dos módulos de entrada, armazena o seu conteúdo numa tabela de dados e executa o programa do usuário armazenado na memória principal, combina as entradas com a lógica de controle, e então atualiza os módulos de saída e sua respectiva tabela de dados.

As combinações de I/O são referenciadas a uma lógica de controle (Programa Principal), que é carregado na memória do PLC (RAM, EPROM ou EEPROM) através de um terminal de programação (PC). Toda essa combinação lógica e seqüencial é ciclicamente executada pela CPU numa ordem pré-determinada, denominada de varredura ou scan. A grande variedade de módulos discretos e analógicos, módulos de comunicação, módulos de rede, diagnóstico de falhas que simplificam a partida e a manutenção do sistema, faz dos PLCs uma excelente opção para projetos de Automação Industrial e até mesmo de Automação Predial.

Um exemplo de uso de PLC, acompanhado de perto por um dos alunos deste projeto, é o sistema utilizado na empresa Samarco Mineração - ES. O PLC usado na Samarco Mineração é o GE FANUC. As considerações mais importantes para a escolha dos PLC's da serie 90-30 GE FANUC pelos engenheiros da empresa Samarco Mineração do Espírito Santo se deve ao baixo custo e o número de pontos de I/O necessários no processo de pelotização. Recursos como a grande variedade de módulos discretos e analógicos, módulos de comunicação, módulos de rede, diagnóstico de falhas que simplificam a partida e a manutenção do sistema, além de possibilitar a integração das outras famílias de PLC's GE FANUC, sendo, portanto, uma excelente opção para projetos de Automação Industrial.

2.2 SDCD

Outra opção do mercado para Automação Industrial é o uso do SDCD (Sistema Digital de Controle Distribuído).

O SDCD é mais potente e robusto que o dos PLC's, principalmente quando se trabalha no tratamento de variáveis analógicas de controle. No geral o SDCD é um sistema proprietário (não é aberto, integrando assim toda parte de supervisão, banco de dados e lógica) que se preocupa em ter um nível de segurança elevado (redundância de comunicação, redes e processamento), o que o faz ter um custo mais elevado se comparado com o PLC. Por outro lado, os PLC's possuem menor nível de redundância, mas por se tratar de um sistema aberto oferece uma maior versatilidade e flexibilidade, permitindo o uso de pacotes (*software*) de terceiros como *In Touch, Windows*, rede *Ethernet* e outros.

A grande desvantagem entre os sistemas que utilizam os PLC's e SDCD's em relação aos que usam os microcontroladores é o elevado custo de implementação e aquisição de PLC's e SDCD's. Em projetos onde não há necessidade de altíssima velocidade de transmissão e recepção (valores menores a 10 ms) deve-se fazer uso de microntroladores que atendam as necessidades do projeto, em virtude do baixíssimo preço de aquisição e instalação dos mesmos.

O SDCD *Bailey INFI 90 OPEN*, por exemplo, é hoje o sistema de controle responsável pelo comando, intertravamento e monitoramento dos principais equipamentos instalados na empresa Samarco Mineração em Ponta Ubu - ES. Um melhor detalhamento deste sistema é apresentado no Apêndice B.

A grande desvantagem entre os sistemas que utilizam os PLCs e SDCDs em relação aos sistemas que usam os microcontroladores é o elevado custo de aquisição e instalação dos PLCs e SDCDs. Em projetos onde não haja necessidade de altíssima velocidade de transmissão e recepção (valores menores a 10ms), deve-se fazer uso de microcontroladores que atendam as necessidades do projeto, em virtude do baixíssimo preço de aquisição e instalação dos mesmos.

2.3 Supervisório

Em um sistema automatizado, muitas vezes é necessária a intervenção, ou somente a visualização de informações, pelo operador do processo, ou, ainda, da seqüência de operação da máquina. Para esse fim são utilizados vários equipamentos, entre os quais podem-se citar:

- Quadro sinótico;
- Software de supervisão, também chamado *Software* Supervisório (SCADA - *Supervisory Control And Data Acquisition System*);
- Interface Homem Máquina (IHM).

Supervisório é um aplicativo que centraliza as informações dos controladores, módulos de aquisição de sinais e outros dispositivos utilizados na automação e controle. Com ele, pode-se monitorar, modificar as lógicas de controle, executar acionamentos à distância, gerar relatórios, visualizar alarmes etc.

Dentre os *software*s disponíveis no mercado, tem-se o *Elipse* SCADA. Este é o atual *software* utilizado no *Shopping* Pátio Brasil, verificado por nós alunos, em visita feita ao Shopping, com apresentação feita pelo gerente de Operações Élsio Augusto Macedo, em maio deste ano. O programa Elipse SCADA funciona, basicamente, pela definição de *tags*, informações de interesse, e pelas operações efetuadas com elas. É possível criar animações, *displays*, relatórios, bateladas, gráficos de tendência, telas gráficas etc. Uma característica muito importante é que o sistema fica aberto para constante ampliação, requisito fundamental para uma automação efetuada pela própria equipe do *shopping*.

Trata-se de um *software* nacional com divulgação mundial. É utilizado para a criação de aplicativos de supervisão e controle de processos nas mais diversas áreas. Totalmente configurável pelo usuário, permite monitorar variáveis em tempo real, através de gráficos e objetos que estão relacionados com as variáveis físicas de campo. Além disso, o usuário pode fazer acionamentos e enviar ou receber informações para os equipamentos de aquisição de dados.

É possível ainda realizar cálculos através de utilização de linguagem de programação, criar bases de dados históricas, relatórios, receitas, e, inclusive, supervisionar e controlar um processo à distância.

Como descrito anteriormente, em um sistema supervisório deve-se monitorar, modificar a lógica de controle, executar acionamentos à distância, gerar relatórios, visualizar alarmes, verificar regiões com funcionamento anormal, modularizar para encontrar erros mais facilmente entre outras características.

Um exemplo de sistema de controle otimizante é o OCS (*Optimizing Control System*) instalado na empresa Samarco Mineração. Maiores detalhes deste sistema de controle encontram-se no Apêndice C.

3 Fundamentos Teóricos: Interfaces de Comunicação de dados

3.1 RS-232C

Antes de 1960 a comunicação de dados compreendia a troca de dados digitais entre um computador central (*mainframe*) e terminais de computador remoto, ou entre dois terminais sem o envolvimento do computador. Tais dispositivos poderiam ser conectados através de linha telefônica, e consequentemente necessitavam de um modem em cada lado para fazer a decodificação dos sinais.

Considerando está situação o comitê conhecido atualmente como "*Electronic Industries Association "* (EIA) criou a interface serial RS-232C (RS – *" Recommended Standard "*), a qual descreve uma padronização de uma interface comum para comunicação de dados entre equipamentos e especifica as tensões, temporizações e funções dos sinais, um protocolo para troca de informações, e as especificações mecânicas.

Os maiores problemas encontrados pelos usuários na utilização da interface RS-232C estão na ausência ou conexão errada de sinais de controle provocando estouros de buffer *("overflow")* ou travamento da comunicação, função incorreta de comunicação para o cabo em uso provocando uma inversão das linhas de transmissão e recepção ou até mesmo a inversão de uma ou mais linhas de controle *("handshaking")*, porém a maioria dos *drivers* são tolerantes aos erros cometidos e os CI's e demais componentes normalmente não se danificam.

3.1.1 Características Elétricas RS-232C

O padrão RS-232C define, atualmente, 2 níveis lógicos. As entradas têm definições diferentes das saídas e os controles têm definições diferentes do que os dados. Para as saídas, o sinal é considerado em condição de marca, ou estado "1", quando a tensão no circuito de transferência, medida no ponto de interface, é menor que –5V e maior que –15V, com relação ao *signal ground* (terra). O sinal é considerado na condição de espaço (*space*), ou estado "0", quando a tensão for maior que +5V e menor que +15V, também com relação ao *signal ground* (terra). A região compreendida entre –5V e +5V é definida como região de transição.

Para as entradas, o sinal é considerado em condição de marca, ou estado "1", quando a tensão no circuit o de transferência, medida no ponto de interface, é menor que –3V e maior que –15V, com relação ao *signal ground* (terra). O sinal é considerado na condição de espaço (*space*), ou estado "0", quando a tensão for maior que +3V e menor que +15V, também com relação ao *signal ground* (terra). A região compreendida entre –3V e +3V é definida como região de transição. A tabela 01 a seguir mostra as principais características elétricas do padrão RS-232C.

Especificações	**Padrão RS232**
Modo de operação	Ponto a ponto
Tipo de transmissão	Full-duplex
Nº total de *Drivers* e *Receivers*	*1 Drivers* e 1 *Receivers*
Comprimento máximo da rede	12 metros
Taxa de transmissão	20 kbps
Faixa de tensão do circuito de saída	± 25 V
Nível de tensão do circuito de saída – 2 nós (com carga)	± 5 V ± 15 V
Nível de tensão do circuito de saída (sem carga)	± 25 V
Impedância do circuito de saída (ohms)	3 k a 7 k
Consumo de corrente em *Tri-State* (*power on*)	N/A
Consumo de corrente em *Tri-State* (*power off*)	± 6 mA a ± 2 V
Faixa de tensão do circuito de entrada	± 15 V
Sensibilidade do circuito de entrada	± 3 V
Impedância do circuito de entrada (ohms)	3 k a 7 k

Tabela 1 : Especificações elétricas do padrão RS-232 [21]

3.2 CAN Bus

CAN, ou *Controller Area Network*, é um barramento de comunicação de dados serial de alta confiabilidade para aplicações em tempo real, feito especialmente para interligar dispositivos "inteligentes" como sensores e atuadores dentro de um sistema ou subsistema.

Redes CAN podem ser usadas como um sistema de comunicação de padrão aberto para microcontroladores e outros dispositivos inteligentes. O barramento CAN foi originalmente desenvolvido para automóveis, porém, seu uso em aplicações industriais tem crescido consideravelmente. Em ambos os casos, o CAN cobre os principais requisitos: possibilidade de operar em um ambiente eletricamente hostil, características de tempo real e facilidade de uso. Alguns usos de CAN têm sido encontrados no campo de engenharia médica, devido a alguns requisitos de segurança e confiabilidade particularmente exigentes. Também pelas mesmas razões CAN tem sido usado em robôs, empilhadeiras e em sistemas de transporte.

3.2.1 Comunicação baseada em mensagens

O protocolo CAN é baseado em mensagens e não em endereços. Com isso, as mensagens não são transmitidas de um nó ao outro baseando-se no endereço. Embutidos dentro da própria mensagem CAN estão a prioridade e os dados transmitidos. Todos os nós no sistema recebem todas as mensagens transmitidas pelo barramento e responderão com um *acknowledge* (reconhecimento) em caso de recebimento positivo. Cada nó deve então decidir se a mensagem deve ser processada ou simplesmente descartada. Desse modo, uma única mensagem pode se destinar a somente um nó em particular, ou a vários nós, dependendo de como a rede foi projetada.

Um recurso muito útil de CAN é a possibilidade de um nó pedir que outro transmita. Essa operação recebe o nome de *Remote Transmit Request* (RTR – Pedido Remoto de Transmissão). Assim, ao invés de ficar esperando que outro nó transmita um dado, o nó pode simplesmente pedir que ele envie a informação. Pode-se tomar como

exemplo o sistema de segurança embarcado em um carro, que recebe atualizações freqüentes de subsistemas críticos como *airbags*, mas não recebe muita informação sobre a tensão da bateria ou a pressão do óleo. Periodicamente o sistema pode pedir que esses sensores enviem dados, de forma que ele possa fazer uma verificação do sistema. Os ganhos na diminuição do tráfego de rede são claros, ao mesmo tempo em que se mantém a integridade e a funcionalidade do sistema.

Outra vantagem desse protocolo baseado em mensagens é a eliminação da necessidade de se reprogramar toda rede no caso de adição de novo nó. Ele simplesmente passa a processar as mensagens de acordo com o identificador (ID) programado para recepção, e passa a enviar mensagens para os nós desejados.

3.3 RS-485

Atualmente, o padrão RS-485 é a rede mais utilizada em ambiente industrial. Este padrão nada mais é do que uma evolução do padrão RS-232C de comunicação serial baseado na transmissão diferencial balanceada de dados, a qual é ideal para transmissão em altas velocidades, longas distâncias e em ambientes sujeitos a interferência eletromagnética, além de permitir a comunicação entre vários elementos participantes em uma mesma rede de dados.

A topologia de utilização do RS-485 é definida como um barramento multiponto, ou seja, vários equipamentos (nós) podem ser ligados à rede ao mesmo tempo (seguindo as especificações). Durante uma comunicação somente um nó poderá enviar dados por vez, isto é, ele tem o "controle" da rede naquele momento. Com isso, enquanto ele envia os dados, todos os outros participantes o recebem e somente o nó endereçado pelo pacote de dados responde ao nó que enviou a requisição. Na prática, cada nó tem o seu endereço ajustável por hardware ou *software* e, ao "ouvir" um pacote de dados enviado pelo nó que tem o controle da rede naquele momento, o nó de destino responde para todos incluindo o que fez a requisição da mensagem, porém, só o controlador interpreta os dados.

O RS-485 é projetado para ser imune a vários tipos de falhas associadas ao ambiente em que está o cabo de comunicação, tais como ruídos elétricos (provenientes de indução magnética) ou até mesmo diferença de potencial entre os terras dos nós da rede,

configurando, assim, uma rede robusta. Este fator é de suma importância para a integridade de um sistema.

Hoje em dia, tem-se tomado ciência do problema de interferência nas indústrias, e tem-se protegido os equipamentos de forma abrangente: pela alimentação, pela comunicação (RS-485), pelos sensores do campo, enfim, blindando o sistema com todos os recursos disponíveis. A rede RS-485 permite em alguns casos distâncias de até 1200 metros entre um terminal e outro.

A taxa de transferência de dados é baixa por se tratar de comunicação serial. Seu uso em sistemas de automação deve-se ao fato de não exigir equipamentos extras para conexão de vários elementos (*hubs, switches*, repetidores etc.). O custo com cabeamento é baixo. Geralmente, é utilizado para ligação entre CLP's e módulos de aquisição de sinais.

Outro fator importante para adoção do protocolo RS-485 é a comunicação de um sistema atual a ser trabalhado com o protocolo RS-232, pois existem no mercado empresas que fazem conversores RS-232/RS-485. Porém não basta somente esta conversão do protocolo elétrico, mas também a alteração no programa que acessa a porta serial RS-232, porque o padrão RS-232 utiliza como modo de operação a transmissão *full-duplex* e comunicação ponto a ponto, enquanto que o padrão RS-485 usa como modo de operação a transmissão *half-duplex*.

Com a necessidade cada vez maior de redes *Fieldbus*, implementadas em indústrias, escritórios e residências, de trocar informações com outras redes, tem-se a necessidade primeira de se utilizar um conversor de protocolo para grandes distâncias, uma maior velocidade de comunicação, e uma rede mais robusta (imune aos ruídos eletromagnéticos) e, para tanto, o protocolo RS-485 é o protocolo elétrico mais utilizado em redes *Fieldbus.* A tabela 02 a seguir faz uma comparação entre os padrões RS-232 e RS-485.

Especificações	Padrão RS232	Padrão RS485
Modo de operação	Ponto a ponto	Multiponto
Tipo de transmissão	Full-duplex	Half-duplex
Nº total de *Drivers* e *Receivers*	1 Drivers e 1 Receivers	1 Drivers e 32 Receivers
Comprimento máximo da rede	12 metros	1200 metros
Taxa de transmissão	20 kbps	10 Mbps
Faixa de tensão do circuito de saída	± 25 V	- 7 V a + 12 V
Nível de tensão do circuito de saída (com carga)	± 5 V ± 15 V	± 1,5 V
Nível de tensão do circuito de saída (sem carga)	± 25 V	± 6 V
Impedância do circuito de saída (ohms)	3 k a 7 k	54
Consumo de corrente máximo em *Tri-State* (*power on*)	N/A	± 100 µA
Consumo de corrente máximo *Tri-State* (*power off*)	± 6 mA a ± 2 V	± 100 µA
Faixa de tensão do circuito de entrada	± 15 V	- 7 V a + 12 V
Sensibilidade do circuito de entrada	± 3 V	± 200 mV
Impedância do circuito de entrada (ohms)	3 k a 7 k	≥10,6 k

Tabela 2: Comparação entre os padrões RS-232 e RS-485 [21]

Definido o meio físico para a conexão dos equipamentos, deve-se analisar qual protocolo será utilizado. Para que o sistema seja passível de ampliações futuras, é importante que se utilizem protocolos abertos.

Diferentemente da interface RS-232 a RS-485 é um tipo de interface serial que utiliza sinais diferenciais balanceados.

O padrão RS-485 permite compartilhar ao mesmo tempo a transmissão e a recepção em uma linha balanceada. O intervalo de tensão de modo comum que o *driver* pode suportar vai de **-7 a + 12 Volts** ainda quando ficam em estado de alta impedância.

4 Fundamentos Teóricos: Linguagem Java

O texto apresentado abaixo é baseado em [17] e [19].

A linguagem Java possibilita o desenvolvimento de programas segundo uma abordagem orientada a objetos. Os programas desenvolvidos segundo esta abordagem são compostos por módulos chamados objetos. Os objetos contêm variáveis, denominadas atributos, e procedimentos, denominados métodos. Os métodos contêm o código responsável pelos serviços providos pelos objetos e os atributos são acessados através dos métodos do objeto (atributos encapsulados).

Em orientação a objetos, um tipo de objeto é chamado de classe (padrões a partir dos quais os objetos são criados). Essas classes são identificadas pela palavra-chave *class* seguida pelo nome da classe.

Existem classes semelhantes (atributos e métodos idênticos), como por exemplo as classes PortaComunicacaoSerial e PortaComunicacaoParalela que podem conter alguns atributos e métodos idênticos, e para se evitar que se escrevam os atributos e métodos tenham que ser digitados na definição de ambas as classes faz-se uso da herança junto a implementação de uma hierarquia de classes.

A linguagem Java apresenta características que facilitam a implementação de aplicações distribuídas. Aplicações distribuídas são uma tendência na computação. Em tais aplicações, os programas são executados em diferentes máquinas e trocam informações através de uma rede de comunicação. A maioria das comunicações distribuídas apresenta uma arquitetura cliente-servidor. Nesta arquitetura o programa cliente solicita serviços ao programa servidor que, após executá-los, envia respostas ao cliente.

Entre as facilidades para a implementação de aplicações com a arquitetura mencionada no parágrafo anterior se destacam as seguintes:

- Alta portabilidade do código;
- Existência de suporte para comunicação através da rede;
- Facilidade para acesso a banco de dados;
- Suporte a múltiplos *threads* de execução.

Na maioria das linguagens, um código executável é composto por instruções específicas da plataforma para a qual foi compilado. Mesmo que o programa utilize uma linguagem padronizada, é necessário que o código seja compilado para cada uma das plataformas onde o programa será executado (incompatibilidade entre o sistema operacional Windows e uma máquina RISC com o sistema operacional UNIX, por exemplo).

Quando um programa Java é compilado, em vez de ser gerado código destinado a uma plataforma específica, é gerado código que pode ser interpretado por uma máquina virtual (alta portabilidade ao código). Qualquer máquina em qualquer plataforma pode executar este código, desde que saiba como interpretar as instruções geradas para esta máquina virtual. Por exemplo, um navegador, como o *Internet Explorer* da *Microsoft* ou o *Navigator* da *Netscape*, sabe interpretar o código Java carregado através da rede.

Esta característica da linguagem Java possibilita que, em uma aplicação cliente-servidor, um mesmo cliente seja interpretado em máquinas com diferentes processadores e sistemas operacionais. Um exemplo claro de aplicações Java que faz uso desta facilidade são as *applets*.

A popularidade da linguagem Java se deve, em grande parte, à esta possibilidade de se implementar programas que são armazenados em servidores e posteriormente transferidos através da rede, com navegadores como o Internet Explorer da Microsoft ou o Navigator da Netscape, para execução nas máquinas dos próprios usuários. Estes programas são chamados de *applets*.

Para que uma applet seja carregada, é necessário criar um arquivo HTML com *tags* apropriadas. Essas *tags* informam ao navegador, responsável por interpretar o código recebido dos servidores através da rede, os endereços onde as *applet*s estão armazenadas.

Além dos *applets*, tem-se os programas *servlets* na linguagem Java que são usados nos servidores Web para processar solicitações recebidas dos clientes em aplicações

distribuídas. O servidor Web é o responsável por intermediar a comunicação entre o cliente e o *servlet* adequado. Uma desvantagem para o uso dos servlets é a necessidade de uma máquina virtual Java no servidor da Web.

Entre as facilidades providas para troca de dados através da rede, pode-se destacar o uso de URLs, o suporte provido pela classe *Socket* e pela chamada a procedimentos remotos (RPC).

A URL (*Uniform Resource Locator*) identifica de forma única um recurso na Internet, devido a sua composição em duas partes em que um identifica o protocolo para acesso ao recurso e outra identifica a localização do recurso.

Aplicações distribuídas usam os serviços de comunicação providos pela rede através de interfaces de programação (APIs). As principais APIs para programação em redes TCP/IP são conhecidas como *Sockets* e ATI. Através das funções presentes nestas APIs é possível o estabelecimento de conexões, o envio e a recepção de dados através da rede de comunicação. Esta comunicação através da rede pode ser orientada a conexão ou baseada em datagramas.

Nas redes TCP/IP a comunicação orientada a conexão utiliza o protocolo TCP enquanto a baseada em datagramas utiliza o protocolo UDP.

A comunicação orientada a conexão é feita através das classes *ServerSocket e Socket*. O servidor usa a classe *ServerSocket* para aguardar conexões a partir dos clientes. Quando ocorre a conexão, a comunicação é efetivada através de um objeto da classe *Socket*. Estas classes escondem a complexidade presente no estabelecimento de uma conexão e no envio de dados através da rede.

Embora o uso de APIs como *Sockets* possa ser a solução adotada em grande quantidade de aplicações, pode-se também usar uma API de mais alto nível conhecida como Chamada a Procedimentos Remotos (RPC – *Remote Procedure Call*). Estes ambientes escondem do programador boa parte dos detalhes envolvidos no uso dos serviços de comunicação providos pela rede como os *Sockets*. As RPCs possibilitam que as aplicações chamem procedimentos executados remotamente como se estivessem chamando procedimentos executados localmente.

Um programa com chamadas a procedimentos remotos é inicialmente analisado por um gerador de código, responsável por substituir as chamadas aos procedimentos remotos

por chamadas a procedimentos locais conhecidos como procedimentos *stub* (inseridos no código da aplicação cliente) e pela inserção de código nas aplicações que serão executadas nos servidores. Este código inserido nas aplicações, executadas pelos servidores, recebe a solicitação do cliente, identifica o procedimento local a ser executado e envia a resposta ao cliente.

A chamada a procedimentos remotos é provida em Java através da API conhecida como *Remote Method Invocation* (RMI). É através da RMI que uma aplicação Java pode chamar métodos de objetos localizados em outras máquinas.

Quando se trabalha com a comunicação cliente-servidor percebe-se que uma grande quantidade de aplicações cliente-servidor precisam acessar informações armazenadas em bancos de dados. O ambiente de desenvolvimento Java provê uma interface de programação padrão para acesso a bancos de dados. A interface de programação é chamada JDBC. Através desta interface é possível estabelecer conexões com servidores de bancos de dados, enviar enunciados SQL e receber resultados. SQL é uma linguagem usada para consultar às informações armazenadas nos bancos de dados.

A última vantagem, citada anteriormente, da linguagem Java para uma rede distribuída é a da linguagem suportar a implementação de aplicações onde seja útil o uso de múltiplos *threads* de execução.

Os múltiplos *threads* de uma aplicação são executados de forma concorrente. No caso de uma máquina que possui apenas um processador, os *threads* compartilham o tempo de execução deste processador, ou seja, enquanto um *thread* encontra-se em execução, os outros estão bloqueados ou aguardando a liberação do processador. Como em programas servidores é desejável que diferentes solicitações de serviço possam ser atendidas simultaneamente. A possibilidade de se implementar aplicações com múltiplos *threads* é bastante útil.

A linguagem Java provê suporte para criação, término e sincronização de *threads*. O suporte disponível na linguagem evita que o programadores tenham que usar interfaces de programação específicas de cada sistema operacional.

5 Fundamentos Teóricos: Sistemas Térmicos

Diversos problemas envolvem o controle de temperatura em fluidos. Controle de temperatura pode ser aplicado a sistemas de ar-condicionado e processos de fabricação com aquecimento ou resfriamento de determinada substância. Estes sistemas podem ser descritos através de parâmetros como “resistências térmicas” e “capacitâncias térmicas”. Estes parâmetros não são concentrados, como em circuitos elétricos, mas devem ser considerados distribuídos para análises mais precisas. Para processos onde se deseja apenas uma estimativa do seu comportamento dinâmico, o modelo usando parâmetros concentrados pode ser utilizado.

Estes sistemas térmicos realizam a transferência de calor entre corpos por meio de três diferentes formas: condução, convecção e radiação. A maioria dos sistemas de controle para processos térmicos envolve condução ou convecção, já que a troca de calor por radiação é considerável apenas quando o corpo emissor está a uma temperatura muito maior do que o corpo receptor.

Para sistemas térmicos, a relação entre fluxo de calor e variação de temperatura pode ser descrita por uma relação diretamente proporcional. A equação (1) demonstra este comportamento, para transferência de calor por convecção:

$$q = K\Delta T = (H.A)\Delta T \qquad (1)$$

Onde:

- ΔT: diferença de temperatura;
- H: coeficiente de convecção;
- A: área normal à convecção.

As grandezas “resistência térmica” e “capacitância térmica” podem ser descritas através das seguintes equações [4]:

$$q = mc\Delta T \tag{2}$$

$$R = \frac{d(\Delta T)}{d(q)} = \frac{1}{K} \tag{3}$$

$$C = \frac{q}{\Delta T} = mc \tag{4}$$

A “capacitância” ou “capacidade” térmica está diretamente relacionada ao calor específico “c” da substância, enquanto que a “resistência térmica” indica qual a oposição que o meio oferece para que o fluxo de calor se propague.

Para controle de temperatura em ambientes, o acionamento pode ser para aquecimento ou resfriamento. Para sistemas que envolvem aumento de temperatura, aquecedores e ventiladores associados são usados para fornecer o fluxo de calor necessário para variar a temperatura da maneira desejada. Para relacionar a variação de temperatura e o fluxo de calor em ambientes, a analogia com circuitos elétricos pode ser novamente usada. Considerando a resistência térmica em série com a capacitância térmica e desejando-se saber qual a temperatura no ambiente, a seguinte relação pode ser demonstrada:

$$\frac{T(s)}{T_q(s)} = \frac{1}{RCs+1} \tag{5}$$

Onde “T(s)” e “Tq(s)” são transformadas de Laplace das variáveis: temperatura do ambiente e temperatura no aquecedor, respectivamente. Esta equação, que corresponde a um sistema de primeira ordem, possui comportamento bem definido e auxilia a análise e determinação experimental dos parâmetros deste modelo de representação do sistema térmico.

5.1 Processo a Controlar – Maquete de um escritório típico

O experimento a controlar é constituído de uma maquete que possui a configuração de um escritório típico com dois secadores de cabelo (representando dois aparelhos de ar-condicionado), um ventilador-exaustor, sensores de temperatura para captura do valor da variável desejada e bolsas de gelo externas ao escritório (túnel 1 e 2) com a finalidade de simular a presença do sol em um dos lados do escritório hipotético. A figura 2 a seguir mostra um esquema estrutural da maquete.

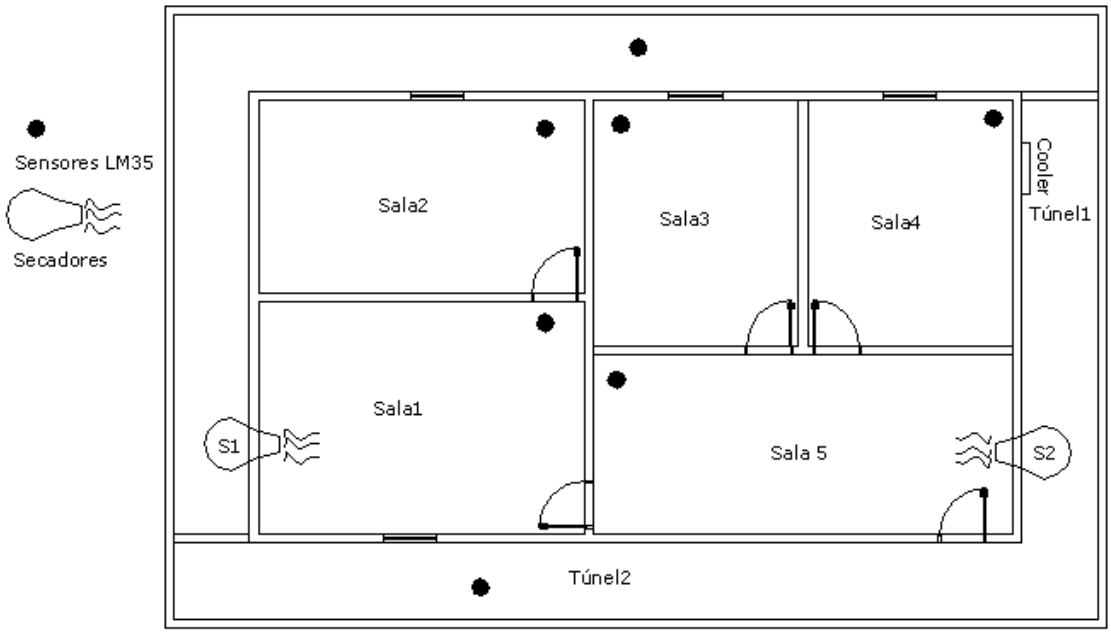

Figura 2: Esquema da maquete utilizada no projeto

À direita temos a sala 5 (1024 cm^2) onde está presente um dos secadores. Ela possui divisões de meia altura que a divide em 3 pequenas salas, com um sensor em cada uma delas. Esse subsistema da maquete permite verificar se existe grande diferença de temperatura entre pontos da mesma sala, lembrando que as divisões de meia altura são apenas “decorativas”.

À esquerda temos a sala 1, menor (608 cm^2), que também possui um secador, podemos verificar a condição de um condicionador de ar muito potente para o tamanho da

sala, e assim poder fazer um controle de uma situação crítica. Vizinha a esta sala existe outra pequena sala (384 cm^2) que possibilita obter os parâmetros da resistência térmica da madeira usada para a confecção da maquete e assim podermos verificar o quão adiabática são as paredes de nosso sistema. Com os valores de resistência é possível modelar todo o sistema, levando em conta que as paredes de todas as salas são de mesmo material.

Cada atuador (secador de cabelo) terá um circuito independente do outro. Circuitos de potência para acionamento e os chips de controle (microcontroladores – PIC16F877) serão independentes para cada secador.

Com a independência dos atuadores no controle do sistema será possível fazer a conexão de cada controlador no barramento que fará a comunicação com o computador servidor e assim tratá-los como experimentos independentes, já que será possível para o usuário remoto fazer a comunicação com cada um dos controladores de forma independente.

6 Fundamentos teóricos: Microcontroladores da Família PIC (Programmable Integrated Controller)

A performance de um microcontrolador depende da sua arquitetura interna, ou seja, do modo em que o microcontrolador foi projetado tanto em hardware como em *software*.

Os dispositivos da família PIC apresentam uma arquitetura alternativa à arquitetura *Von-Neumann*, chamada de *Harvard*. Nesta, existem dois barramentos internos, um para dados e o outro para instruções, diferentemente da *Von-Neumann* que usa apenas um barramento usado tanto para instruções quanto para dados. Esta diferença permite que a CPU possa acessar a memória de dados e a memória de programas ao mesmo tempo. O barramento de dados é de 8 bits, enquanto que o de instruções pode ser de 12, 14 ou 16 bits.

Todos os microcontroladores PIC seguem a tecnologia RISC (*Reduced Instruction Set Computer*), com conjunto de instruções reduzido (em torno de 35 instruções distintas, variando entre os modelos de PIC). As principais vantagens desta filosofia em relação a filosofia CISC (*Complex Instruction Set Computer*) é a facilidade de aprendizado, menor quantidade de instruções, execução otimizada de chamadas de funções, pipeline melhorado, presença de múltiplos conjuntos de registradores, maior segurança e confiabilidade para as operações e melhor desempenho. Como desvantagem tem-se um maior custo por se tratar de uma tecnologia mais recente e eficiente e uma possível dependência de fabricantes de hardware e *software*.

Com menor quantidade de instruções e com cada uma delas tendo sua execução otimizada, o sistema deve produzir seus resultados com melhor desempenho, mesmo

considerando-se que uma menor quantidade de instruções vai conduzir a programas um pouco mais longos.

Assim como outros microcontroladores, o PIC é um circuito dinâmico, e para seu funcionamento necessita de um sinal de relógio para sincronização de suas operações. Este sinal de "clock" é gerado por um oscilador que fornece uma seqüência ininterrupta de pulsos com períodos constantes.

O sinal de relógio nestes componentes é internamente dividido por quatro. Esta divisão se dá em 4 fases a cada ciclo de máquina. Na primeira fase, incrementa-se o contador de programa e busca-se a instrução seguinte da memória de programa; na última fase a instrução é armazenada no registrador de instruções. No próximo ciclo de máquina, entre a primeira e a última fase a instrução é decodificada e executada.

6.1 Modelo de microcontrolador utilizado: PIC16F877A

6.1.1 A estrutura interna

A seguir apresenta-se a estrutura interna deste microcontrolador através de um diagrama de blocos [13] mostrado a seguir na figura 3.

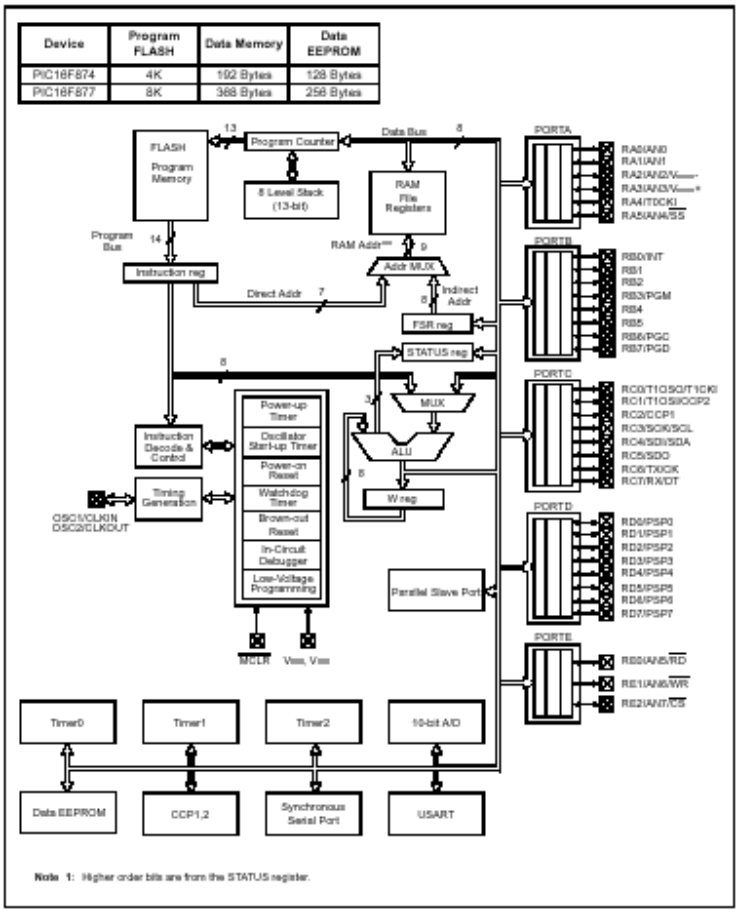

Figura 3: Estrutura interna do PIC16F877A

O diagrama de blocos anterior, detalha todos os periféricos e comunicações que compõe este poderoso microcontrolador. Neste diagrama (figura3) as diversas partes que compõe o microcontrolador PIC16F877A podem ser visualizadas. No centro encontramos a ULA (Unidade Lógica Aritmética), que é a unidade de processamento e está diretamente ligada ao registrador *Work* (W reg). No canto superior esquerdo temos a memória de programa (FLASH). Saindo deste bloco temos um barramento de 14 *bits (Program Bus)*. Mais a direita encontra-se a memória de dados (RAM), a qual possui um barramento de 8 bits (*Data Bus*). Totalmente a direita encontram-se os PORTs (PORTA, PORTB, PORTC, PORTD, PORTE). Na parte inferior tem-se os demais periféricos, tais como: a EEPROM, (memória de dados não volátil), os *timers* (TMR0, TMR1, TMR2), os A/Ds de 10 *bits*, os modos CCP (*Compare, Capture e PWM*) e as comunicações seriais (SPI, I^2C e USART).

Algo a ser observado é que entre todos os periféricos a comunicação é feita através de um barramento de 8 *bits*.

Um pouco mais ao centro da figura 3, podemos encontrar ainda o registrador de status (*STATUS reg*), onde algumas informações importantes sobre a ULA ficam armazenadas, e os demais SFRs (*Special Function Registers*). Na parte superior temos ainda o contador "de linhas" de programa (*Program Counter*) e a pilha de 8 níveis (*Stack*).

Por fim, tem-se ainda os circuitos internos de *reset*, *Power-on Reset* (POR), *Brown-Out Reset* (BOR), osciladores, *Watchdog Timer* (WDT) e sistemas de programação. As demais características do microcontrolador são apresentadas no Apêndice H.

7 Metodologia

7.1 Escolha da arquitetura do laboratório remoto

A escolha da arquitetura do laboratório remoto foi facilitada, em partes, com o auxilio do trabalho já realizado pelo aluno Alexandre Silva Souza em seu projeto de PIBIC sobre Automação de Processo de Tanques Acoplados (Plano de Trabalho – " Sistemas de Acesso Remoto para Controle de Nível em Processo de Tanques Acoplados, 2002 "), no qual foi utilizada a linguagem Java para comunicação entre o usuário remoto e o computador servidor.

A estrutura de comunicação, do então projeto de PIBIC, é baseada numa arquitetura Cliente/Servidor escrita em Java. O usuário pode trabalhar em qualquer plataforma que atende um *Web browser* com um *Java Runtime Enviroment*. O *Web browser* local contém a interface para o experimento. O *browser* carrega o programa cliente com Java *applets* do Servidor e o inicia. O Servidor de comunicação roda em um PC com o sistema operacional *Windows 2000 Professional*. O Servidor *HTTP* IIS supre os documentos HTML e os Java *applet*s.

Ficou definido que a estrutura de comunicação entre o usuário e o computador servidor será a mesma da definida anteriormente, ou seja, através de *softwares* escritos na linguagem Java.

A grande diferença entre o projeto desenvolvido anteriormente e o atual encontra-se na estrutura de comunicação entre o computar servidor e no fato dos processos a serem controlados. Naquele, o computador servidor apresenta um modelo de sistema dedicado figura 4, ou seja, um único experimento associado ao servidor *WEB.* No presente projeto a idéia é que o computador servidor comunique com vários processos presentes e que

possibilite a comunicação com novos experimentos, que venham a ser desenvolvidos, sem maiores dificuldades.

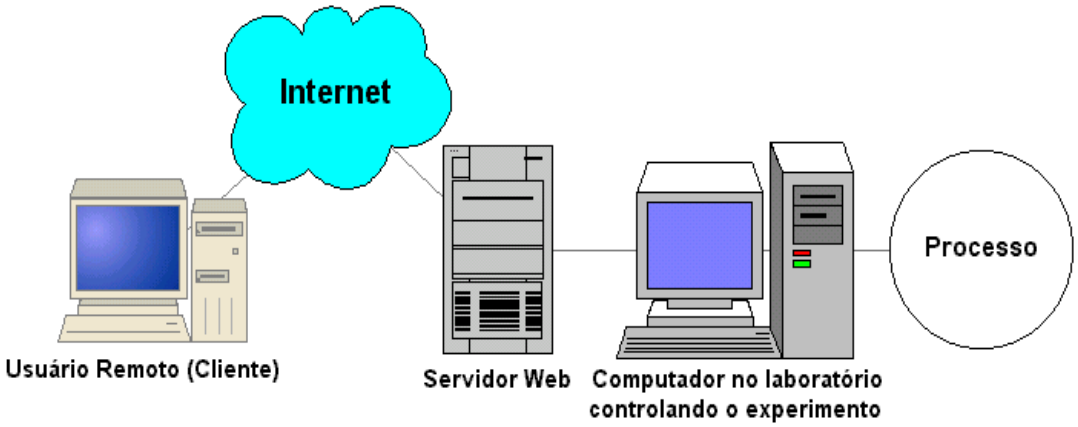

Figura 4: Topologia de um laboratório remoto com Servidor *WEB* dedicado.

Outra grande diferença entre os projetos é que no presente projeto o controlador de processo é um microcontrolador da empresa Microchip e no outro o controle do processo é feita através de um PC com sistema operacional DOS 6.22 que executa o programa de controle escrito na linguagem C. A intenção é que o presente projeto seja flexível de modo que o computador servidor se comunique tanto com microcontroladores como com PC's.

Desta maneira, a próxima etapa de estudo foi verificar qual seria a forma mais viável de comunicação entre os microcontroladores PIC e o computador servidor. A primeira análise foi através do barramento CAN (*Control Área Network*) que é implementado na família de microcontroladores PIC18F. Outra alternativa verificada foi fazer a comunicação através da interface USART que é implementada na maioria dos microcontroladores da Microchip.

As principais fontes de pesquisa nesta etapa do processo foram os sites de busca na internet, em especial o Google [1]. A partir do Google encontramos os sites, ref. [2], [3] e [4], com um rico conteúdo a respeito das principais características do CAN tanto a nível físico (características elétricas) quanto em nível de protocolo.

Para que a comunicação entre o computador servidor e o microcontrolador PIC fosse realizada diretamente através do protocolo CAN precisa-se de uma placa para PC que implementasse o protocolo CAN. Para isso investigou-se todos os atributos relacionados a

esta placa através da ref. [4] e [5]. As características procuradas foram encontradas na referida placa, no entanto, o preço comercial da mesma apresentou-se alto (U$400, quatrocentos dólares) para o presente projeto.

Outro fator que se apresentou de forma negativa para a implementação da comunicação direta CAN foi à necessidade de utilizar os microcontroladores da família PIC18f que são mais caros e difíceis de encontrar.

Após verificar a possibilidade de comunicação direta CAN, investigou-se a outra forma de comunicação com o PIC que é através de sua porta serial (Interface USART). Para isso, estudou-se, através da ref. [6] e [7], a forma como o PIC implementa esta comunicação. Seguindo este mesmo padrão de comunicação, estudou-se a forma como os PC´s atuais fazem a comunicação através da interface serial. As fontes de documentação utilizada nesta fase foram alguns sites da internet e, em especial, o livro da ref. [8].

A documentação [8] trata da comunicação, através da porta serial do PC, utilizando o padrão RS-232 e a linguagem de programação “C”. Porém, como descrito anteriormente, a linguagem que será utilizada no computador servidor para fazer a comunicação com o usuário remoto será Java. Sendo assim, uma solução bastante interessante seria fazer também a comunicação com a porta serial através da linguagem Java, pois teríamos apenas um programa comunicando tanto com o usuário remoto quanto com o microcontrolador.

Para verificar a possibilidade de fazer a comunicação com a porta serial do PC através da linguagem Java o conteúdo da ref. [9] foi exaustivamente consultado. A partir desta referência verificamos que é possível e bastante viável utilizar a linguagem Java para fazer a comunicação serial. A *Sun* disponibiliza gratuitamente uma API (*Applications Programming Interface*) completa de comunicação com a porta serial, a *javax.comm*.

Dessa maneira vimos que seria possível fazermos a comunicação serial utilizando o padrão RS-232 entre o computador servidor e o microntrolador. Porém essa comunicação é ponto-a-ponto e, assim, a priori permite a conexão de apenas um elemento à porta serial. Com essa limitação decidimos manter esse padrão de comunicação tanto no computador servidor como no microcontrolador e então fazer uma conversão de padrões de modo a construir um barramento que possibilitasse a conexão de vários elementos, ou seja, vários processos.

Em [10] estudamos o padrão de comunicação RS-485 que permite a criação de um barramento e, assim, a possibilidade de conexão de vários elementos. Para implementar esse padrão no barramento, necessitaríamos de um conversor LTC491 para converter o padrão RS-232 para RS-485 e vice-versa. A figura 8 mostra o esquema de comunicação através deste barramento.

A partir de pesquisas nos sites verificamos que também é possível, [11], a criação de um barramento que utiliza o padrão CAN. Neste caso necessitaríamos de vários componentes como o *transceiver* PCA82C251 da Philips e o controlador de barramento MCP2510 da Microchip. A figura 9 mostra o esquema de comunicação através deste barramento.

Através de [12], descobriu-se uma maneira de implementar um barramento utilizando o próprio padrão RS-232 e dessa maneira não necessitaria de fazer conversões de barramento. A figura 7 mostra o esquema que pode ser utilizado para implementar tal barramento.

O esquema do barramento RS-232C da figura 7 impõe que apenas o computador servidor (Mestre) pode fazer a comunicação com os processos, ou seja, os processos não podem fazer comunicações entre si. No entanto, esta restrição não prejudica em nada nosso projeto já que queremos apenas fazer a comunicação entre o computador servidor e os demais elementos pertencentes ao barramento.

Depois de uma análise cuidadosa decidimos que o barramento a ser implemento na arquitetura de comunicação do laboratório remoto seria o RS-232C. A principal vantagem que encontramos nesse barramento foi a simplicidade de implementação quando comparado tanto com o barramento RS-485 como com o CAN. Outra vantagem é no tipo de transmissão que neste é do tipo *Full Duplex* enquanto que no padrão RS-485 é do tipo *Half Duplex*. Uma compilação de todas as características do padrão RS-232C juntamente com sua simplicidade de implementação fará com que o presente projeto venha a ter uma arquitetura de comunicação interna eficiente, robusta e com baixo custo.

7.2 O processo Térmico

Para estudar um sistema de controle de ar condicionado é necessário que tenha-se espaço disponível para realizar os experimentos, ou seja, precisaria de, pelo menos, uma sala onde seriam feitas as medições e onde o ar condicionado iria atuar. Se pensar num sistema maior, com mais aparelhos de ar condicionado e mais salas teríamos um alto custo de manter este número de salas inativas, utilizando um espaço que a universidade não tem disponível.

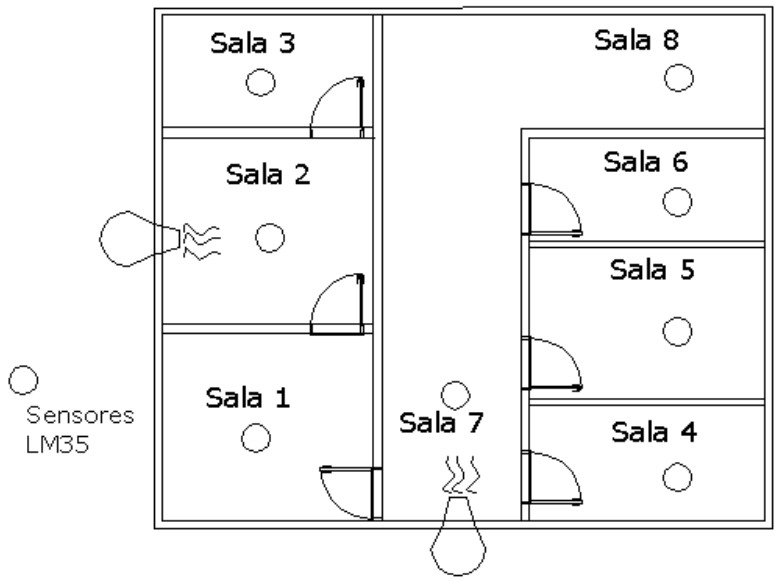

Figura 5: Maquete do Sistema Térmico antiga

Inicialmente foi utilizada a maquete mostrada na figura 5. Mas ela possuía um grande problema, os sensores das salas que possuíam secador ficavam muito próximos ao mesmo, fazendo com que a temperatura nesses pontos rapidamente chegassem ao ápice, dificultando, assim, a análise. Outro problema era o elevado número de salas.

A saída utilizada foi reduzir o sistema, de forma que fosse possível tê-lo em cima de uma mesa. Para isso foi confeccionada uma nova maquete, imitando um escritório, com diversas salas e dois secadores de cabelo simulando os condicionadores de ar.

A nova maquete em questão foi criada pelos alunos Ênio Salgado Pereira e Antônio Augusto C. Leite os quais fazem parte de um projeto final que estuda os diferentes tipos de

controle em sistemas térmicos [22] e que está sendo realizado em paralelo com o desenvolvido por nós.

7.3 Escolha e compra do Microcontrolador

O custo de um microcontrolador hoje "é inferior" comparado a qualquer circuito analógico com as mesmas funções. Em outras palavras, desenvolver um circuito "discreto" utilizando CI´s, transistores, etc, etc, além de ser "mais caro", torna-se imutável para futuras alterações. Alterar a estrutura de um hardware significa mudar PCI (placa de circuito impresso), mudar muitas vezes CI´s, circuitos, etc. Alterar o desempenho de um microcontrolador (funções) significa em resumo trabalhar com o a linguagem de programação do mesmo (acrescentar ou retirar instruções).

Tendo como principal fonte de informações a *internet* e o projeto anteriormente realizado pelo ex-aluno Fernando de Melo Luna Filho [23], fez-se um estudo comparativo de modelos de microcontroladores encontrados no mercado e decidiu-se pelo uso do microcontrolador PIC16F877A. O que pesou para está decisão foi o fato deste modelo de microcontrolador ser semelhante ao utilizado em [23], mas com algumas melhorias. Além disso, este componente, como anteriormente citado, possui inúmeras vantagens para o controle de processos térmicos e construção de um sistema supervisório como: três contadores (*timers*), dois módulos que servem para captura, comparação e geração de sinal PWM, conversor analógico-digital embutido, uso de até 33 pinos para operações de entrada e saída, módulo interno destinado à comunicação serial no modo síncrono e assíncrono.

Outros fatores determinantes para aquisição do referido equipamento foi o seu baixo custo (R$ 27,50) associado à possibilidade de um serviço de assistência técnica, menor custo de manutenção, grande quantidade de documentos sobre o uso do microcontrolador, simplicidade de utilização (arquitetura RISC) e uso dos trabalhos já realizados pelo ex-aluno Fernando na UnB.

Pesquisas feitas com outros microcontroladores como o 8051 e outros da sua família, os microcontroladores da *Motorola* e tantos outros revelaram o menor custo benefício dos microcontroladores da família PIC16F em relação às necessidades do projeto como um todo.

Na teoria tudo isso era perfeito. Porém ao iniciar o uso do microcontrolador constatou-se que apesar do mesmo ser um *up-grade* do microcontrolador PIC16F877 não havia uma biblioteca no *software* de compilação HPDPIC^{TM} que utiliza-se da programação em linguagem C e convertesse o código fonte desenvolvido, em código hexadecimal. Este código seria então utilizado no *software* de gravação ProgPIC. Caso este *software* possuísse uma biblioteca para o PIC16F877A, poder-se-ia utilizar a opção de fazer o código em MPASM e usar o circuito de gravação já pronto e utilizado em [23]. Infelizmente não havia tal biblioteca também.

Deparou-se então com um grande problema: a necessidade de comprar um gravador e um *software* de gravação. Após inúmeras pesquisas na *internet* sobre gravadores e *softwares* de gravação escolheu-se o gravador *µC Flash* da empresa Mosaico, por se tratar de um gravador eficiente e de baixo custo frente aos demais (R$ 99,00) e o *software* MPLAB versão 5.70.40 por se tratar de uma ferramenta da Microchip que é padrão de comunicação com o *µC Flash* que além de editor, permite simular o programa efetuado e gerar o arquivo HEX para a programação do PIC no próprio *software*. O único problema encontrado foi à impossibilidade de programação em mais alto nível com o uso da linguagem C, por exemplo, pelo menos para os microcontroladores da família PIC16Fxx. É válido ressaltar que a partir dos PIC's da família 17F e 18F já existem ferramentas do próprio MPLAB que possibilitam a edição e compilação de programas desenvolvidos na linguagem C. Portanto, havia manuais a disposição via *internet*, assistência técnica oferecida pelos fabricantes da placa (Mosaico Engenharia) com o circuito de gravação, e um *software* mais completo e sem custos.

Por último e não menos importante, cogitou-se a possibilidade de trabalharmos com o PIC18F458 que possui embutido o barramento CAN Bus e outras inúmeras vantagens no processamento. Porém, a dificuldade de obtenção do PIC, apenas nos E.U.A., e a pouca quantidade de documentos sobre o uso do mesmo, contribuíram para a não utilização do PIC 18F458.

7.4 Procedimentos para uso do PIC16F877A

O microcontrolador PIC pode ser programado através da escrita em sua memória de programa. Esta gravação do dispositivo é feita com o uso do programador "MPLAB IDE".

O dispositivo pode ser programado usando-se linguagem de programação *assembly*, "MPASM". O PIC, sendo um processador que segue a tecnologia RISC, possui apenas 35 instruções básicas. Qualquer outra operação a ser efetuada pelo PIC que não pertença a este conjunto de instruções deve ser programada, inclusive operações com números do tipo ponto flutuante. Esta propriedade dificulta sua programação em *assembly*, pois operações em variáveis do tipo ponto flutuante serão utilizadas nos algoritmos de controle projetados.

Após a correta montagem, o arquivo que será usado para gravar o PIC é o arquivo com extensão "hex" produzido após a compilação. Este arquivo contém a seqüência de código binário em linguagem de máquina do microcontrolador que será executada.

O *software* MPLAB para gravação do PIC possui interface amigável, tornando a tarefa de programação dos microcontroladores mais fácil. Este *software* também possibilita leitura do programa presente em um microcontrolador qualquer, bem como a seleção de opções de configuração para gravação, configuração da porta de comunicação que será usada no computador, simulação da programação realizada (tempo, registradores, código passo a passo etc).

Com este ambiente de programação, também se torna possível construir bibliotecas (formato .inc) para uso dos módulos embutidos do PIC, o que facilita a sua programação e modulariza os programas.

Seguindo o "Guia do Usuário - Gravador *µCFlash"* que vem junto com o gravador instalou-se o MPLAB, configurando a porta serial que seria utilizada para a gravação, habilitou-se o *µC Flash* selecionando "*PICSTART Plus / Enable*" (com o gravador já conectado com a porta serial (DB9) do computador), selecionou-se o microcontrolador desejado e checaram-se os bits de configuração.

Realizada a instalação do *software* e a configuração do mesmo, partiu-se para a programação. Primeiramente, fez-se um programa simplório para testar a comunicação serial entre o PIC e o PC visando dar início ao aprendizado para a construção do supervisório. O programa é mostrado no Apêndice A.

Ao término do programa apresentado acima se realizaram simulações. Ao fim das simulações e já com o circuito gravador conectado ao computador procedeu-se a gravação do programa serial.asm no PIC selecionando o "PICSTARTPlus / Enable", fazendo as devidas configurações, checando se o PIC está em branco e selecionando o icone "Program" para realizar a gravação. Os demais programas utilizaram basicamente o mesmo procedimento. Não cabe aqui apresentar todos os programas feitos no MPASM, mas sim mostrar a parte lógica de cada programa. No item "Resultados Obtidos" serão apresentadas as partes lógicas de cada programa utilizado no projeto.

7.5 Procedimento para utilização da porta serial

No intuito de desenvolver o *software* que faça a comunicação de dados com o microntrolador PIC utilizando a porta serial do PC, foi desenvolvido inicialmente um programa em java que comunica com outro PC através da porta serial. Este programa foi criado como uma metodologia para desenvolver o programa principal que faria a comunicação com o PIC. Comunicando com outro PC seria possível fazer comunicação através de mensagens, que possibilitariam inspeção visual, pois estaríamos vendo através de uma interface as mensagens trocadas, e assim validar a integridade do *software*.

Para isso foi desenvolvido o cabo que conectou os PC´s. O cabo constitui-se de dois conectores DB9-fêmea e um cabo de rede.

Após desenvolver o *software* tivemos dificuldade em obter a porta para comunicação em um dos PC´s (sistema operacional windows 2000 professional). Mesmo não tendo nenhum hardware conectado no conector do PC o programa respondia que a porta estava em uso e assim não poderia obtê-la para transmissão. Como o PC possuía apenas esta porta serial, COM1, e não sabíamos o porquê do sistema acusar que a mesma estava em uso resolvemos então alterar o número da porta para COM2 (*Painel de controle → Sistema → Hardware → Gerenciador de Sistema → Propriedade da Porta comunicação COM1 → Configuração → Avançado*). Realizado este procedimento o sistema operacional respondeu que a porta COM1 estava sendo usada por algum aplicativo, mas mesmo assim possibilitou a mudança. Desta maneira alteramos o programa Java para

tentar abrir a porta COM2 e não mais a COM1. Com isso o programa não teve mais problema na obtenção da porta para comunicação.

A partir do momento em que os PC´s estavam comunicando observamos que em um dos PC's (Sistema Opeacional Windows 98) as mensagens com mais de oito caracteres eram divididas em mensagens de oito bytes (caracteres) e no outro (Sistema Operacional Windows 2000) as mensagens eram divididas em 14 caracteres. No entanto, o problema maior estava no fato de que nem sempre as mensagens do PC com windows 98 eram divididas em 8 bytes às vezes apareciam mensagens com mais de 8 caracteres.

A questão da diferença no número de bytes em que as mensagens eram divididas nos PC´s foi identificada como sendo configurações diferentes dos buffers de recepção e transmissão das portas serias dos diferentes computadores. Porém o fato de que em alguns momentos, as mensagens não estavam sendo divididas de tamanho fixo não era justificado. A solução encontrada para não repetir este inconveniente foi configurar o buffer de recepção do PC com windows 98 no máximo, ou seja, para 14 bytes. Com isso, este PC passou a dividir as mensagens que recebiam em 14 caracteres de maneira fixa.

8 Resultados Obtidos

Os resultados obtidos no presente projeto foram divididos nos tópicos:

- Arquitetura de comunicação de um laboratório remoto
  - Comunicação Processos ←→ WEB *Server*
  - Comunicação WEB *Server* ←→ Usuário Remoto
- Programação Java;
- Programação Assembly para microcontroladores;

8.1 Arquitetura de comunicação para implementação de um laboratório remoto via internet

Laboratórios remotos podem apresentar um ou mais experimentos *on-line*. A figura 6, a seguir, apresenta um modelo simplificado de sistema para uma quantidade arbitrária de experimentos associados ao servidor WEB. Os vários tipos de estações clientes ilustram a necessidade de construir um laboratório remoto compatível com diferentes plataformas de *hardware* do cliente. Ambos os modelos partem do princípio da utilização de um único servidor WEB. Obviamente, o número arbitrário de experimentos é limitado pela capacidade do servidor WEB, tanto em processamento como em banda passante da rede.

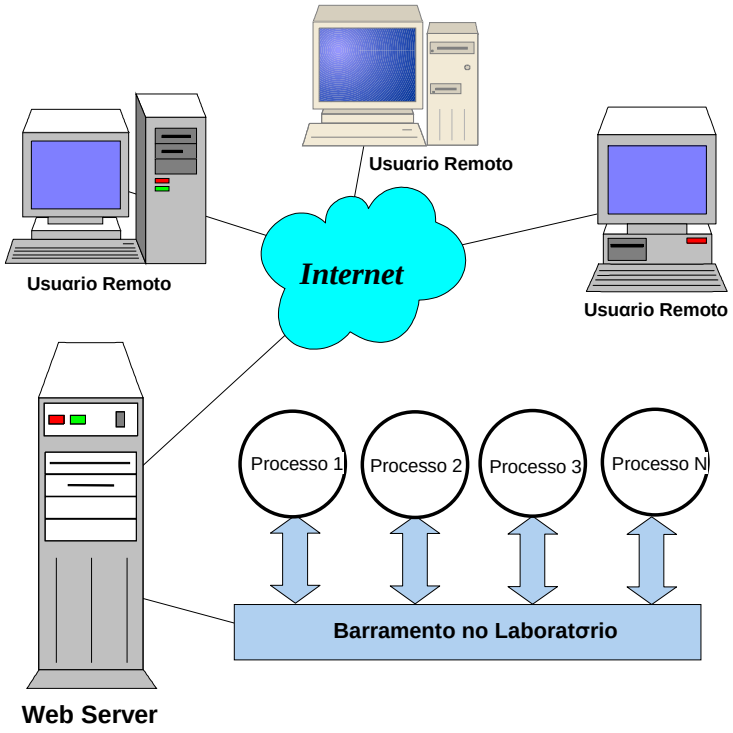

Figura 6: Arquitetura de comunicação de um laboratório remoto.

Para a implementação do laboratório remoto duas arquiteturas devem ser implementadas. Umas delas é a forma de comunicação entre o computador que publica as páginas na internet e os experimentos disponibilizados pelo mesmo. A outra consiste na

forma de interação entre os usuários remotos e o computador que comunica com os experimentos.

Para as arquiteturas citadas anteriormente, algumas características devem ser estabelecidas de maneira a culminar em um projeto que venha a ser, realmente, um laboratório remoto que atenda às necessidades de usuários. Para a comunicação entre usuários e servidor, destacamos:

- ✓ ***Velocidade:*** A comunicação deve ser rápida o suficiente para garantir a troca de informações em tempo real;
- ✓ ***Atualização do sistema:*** Arquitetura auto-suficiente, ou seja, a atualização do sistema não deve depender do usuário e toda modificação deve ser imediatamente disponibilizada para o mesmo.
- ✓ ***Robustez:*** Ferramentas computacionais de proteção contra qualquer eventual dano físico ou lógico do sistema.
- ✓ ***Baixo custo:*** Utilização de *softwares* livres e/ou propriamente desenvolvidos.
- ✓ ***Eficiência Educacional:*** Promover alta interatividade/iniciativa do usuário, facilidade de uso, oferecer informações técnicas sobre o experimento, permitir aprendizagem modular, etc.

Para a comunicação entre o computador servidor e os processos, destacamos:

- ✓ ***Modularidade:*** Permitir a conexão de experimentos com diferentes controladores (microntroladores ou mesmo PC´s);
- ✓ ***Alta portabilidade:*** Permitir a conexão de novos experimentos com o emprego mínimo de novos recursos;
- ✓ ***Velocidade:*** Para garantir a velocidade do laboratório como um todo;

8.1.1 Arquitetura de comunicação entre usuários e computador Web *Server*

O principal serviço da *internet* que impulsionou o ensino a distância é o HTTP (*HyperText Transfer Protocol*), pois permite uma vasta publicação/atualização de documentos, imagens, dados, vídeo, etc. Este serviço vem sofrendo implementações ao longo do tempo e tende a oferecer ao usuário final cada vez mais recursos com o mínimo de *software* e processamento local.

A arquitetura de comunicação é baseada na arquitetura cliente/servidor e é basicamente desenvolvida em linguagem Java. Dessa forma, os estudantes podem utilizar o sistema em qualquer plataforma que suporte um WEB *Browser* com máquina virtual Java. Todos os WEB *Browsers* relativamente recentes já possuem esta máquina virtual. De acordo com a pesquisa realizada, a utilização da linguagem Java é atualmente praticamente uma unanimidade em laboratórios remotos com fins educacionais. Ela possui todas as características favoráveis para o desenvolvimento do *software* cliente bem como as ferramentas necessárias para estabelecer comunicação em tempo real na arquitetura cliente/servidor. É uma linguagem altamente portátil, vastamente documentada, possui bibliotecas matemáticas e acima de tudo, é gratuita. Em geral, *Applets* Java são utilizados nas máquinas clientes para estabelecer comunicação em tempo real com o *software Servidor* Java no servidor WEB. Do ponto de vista da segurança também há vantagem, uma vez que programas escritos em linguagem Java, executando sobre a máquina virtual do WEB *Browser* cliente, não possuem permissões de acesso a dados locais, logo, não é capaz de vasculhar informações do computador cliente sem que a iniciativa tenha sido do próprio cliente.

Localmente, o WEB *Browser* é a interface para o experimento. O WEB *Browser* efetua o *download* do *software* cliente em formato de *Applets* Java a partir do servidor WEB e os executa. O servidor WEB provê a página HTML e os *Applets* Java. Atualmente, é comum encontrar no desenvolvimento de páginas a utilização de linguagens como PHP (*Pré-Hypertex-Processor*), *ASP* (*Active Server Page*) e *JavaScripts*, onde a vantagem destas ferramentas é elevar as possibilidades de análise e processamento dos dados. Em especial, tanto PHP quanto ASP executam seus scripts no próprio servidor WEB, enviando

assim apenas o resultado da operação solicitada pelo cliente. Dessa forma, o processamento local necessário para o computador do cliente *internet* é reduzida.

Applets Java facilitam atualizações no *software*, pois um novo *download* é efetivado a cada requisição de leitura do servidor WEB. Conseqüentemente, do ponto de vista do usuário, nenhuma programação e/ou instalação de *softwares* é necessária para qualquer atualização do sistema.

8.1.2 Arquitetura de comunicação entre processos e o computador WEB *Server*

A seguir são apresentadas algumas possibilidades de se conectar vários processos experimentais, controlados por microcontroladores ou mesmo por computadores, através de um barramento ligado ao computador WEB *Server*.

8.1.2.1 Esquema de ligação através do barramento RS-232C

O padrão RS-232C define o modo de operação ponto-a-ponto. No entanto, é possível construir um barramento no qual vários elementos podem ser conectados, e então, usufruir de uma mesma interface, como também, de um mesmo conjunto de fios tanto para transmitir como para receber dados.

Esta implementação é possível desde que se tenha apenas um "mestre", computador, o qual pode transmitir, como também, fazer requisições de dados a outros elementos.

A figura 7 a seguir mostra o esquema de como as conexões podem ser feitas para implementar o barramento RS-232C "multiponto".

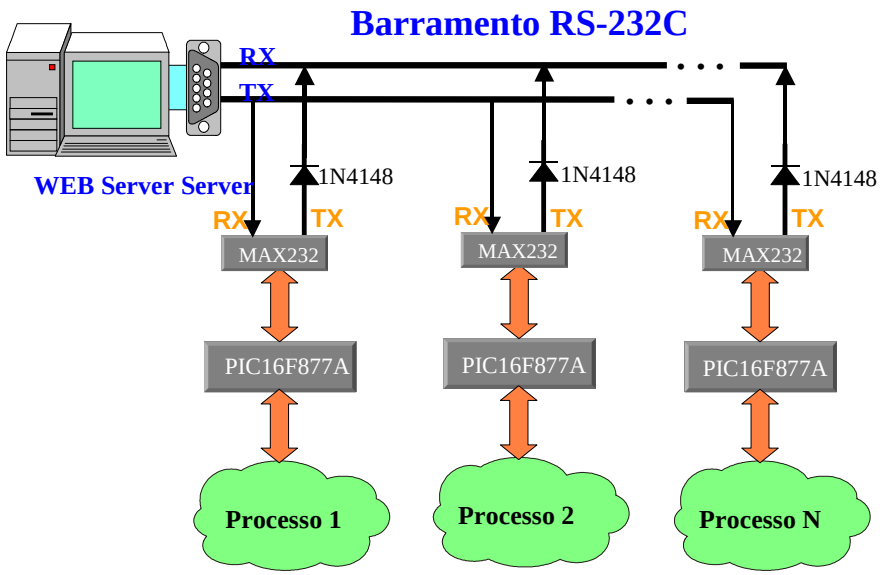

Figura 7: Esquema de ligação do barramento RS-232C

No esquema mostrado, tem-se a necessidade de em cada entidade “escrava” conectar um diodo em sua linha de transmissão. Este elemento é necessário devido à própria característica do padrão RS-232 que define a tensão de –15V para as portas de transmissão quando as mesmas não estão transmitindo. Desta maneira quando uma entidade de um processo qualquer for transmitir dados para o computador evita-se que a corrente atinja as outras entidades “escravas” e dessa maneira prejudique a comunicação entre a entidade transmissora e a receptora.

O MAX232 é um conversor popular para ligação entre circuitos TTL e RS-232. Ou seja, ele converte o padrão elétrico 0V e 5V do PIC para o padrão de tensão –10V e +10V do RS-232 e vice-versa.

8.1.2.2 Esquema de ligação através do barramento RS-485

O esquema a seguir mostra as várias unidades necessárias para interligação de vários PIC´s através do barramento RS-485.

O computador servidor de internet pode comunicar com qualquer controlador de processo, no caso os PIC's, de forma a obter dados relativos ao processo, como também, definir parâmetros de controle e configurar dados relativos ao processo. Tudo em tempo real.

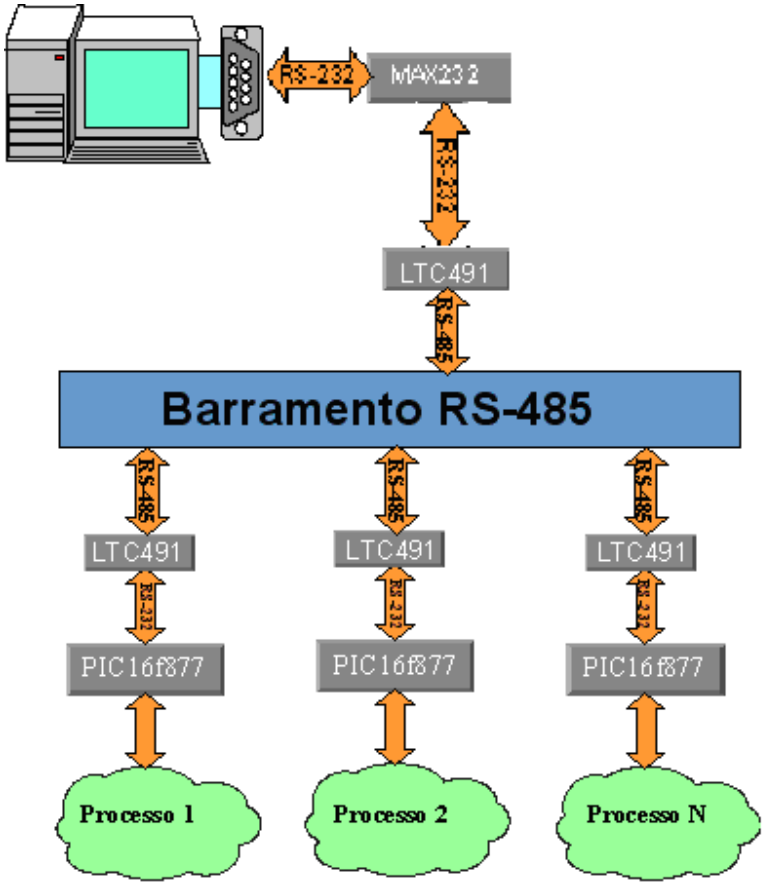

Figura 8: Esquema de comunicação através de barramento RS-485.

O LTC491 é um conversor de baixa potência de padrão RS-232C para RS-485 (e vice-versa). Outra função do LTC é fazer o controle do barramento, ou seja, permite que uma determinada linha possua a permissão de utilização do barramento.

8.1.2.3 Esquema de ligação através do barramento CAN

O esquema a seguir mostra as várias unidades necessárias para interligação de vários PIC´s através do barramento CAN.

O computador servidor de internet pode comunicar com qualquer controlador de processo, no caso os PIC's, de forma a obter dados relativos ao processo, como também, definir parâmetros de controle e configurar dados relativos ao processo. Tudo em tempo real.

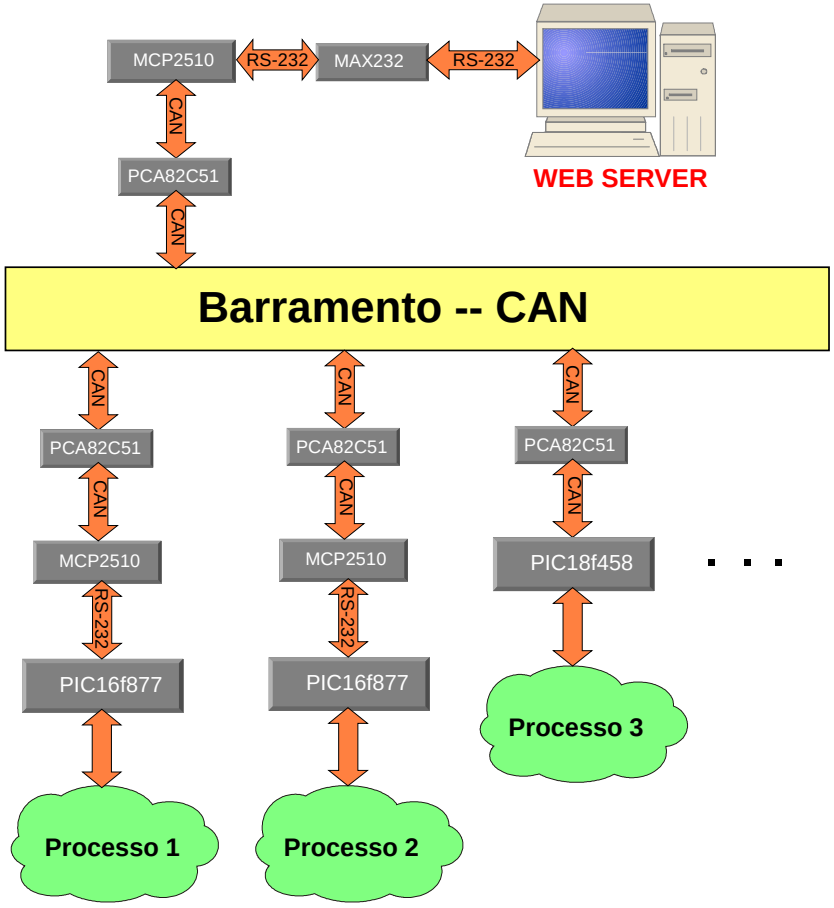

Figura 9: Esquema de ligação de vários PIC´s através de um barramento CAN Bus

O MCP2510 é um controlador de barramento CAN, da Microchip, com interface SPI. Através dessa interface pode-se fazer a comunicação entre o PIC e o MCP com taxas de dados que podem alcançar 5 Mbps.

Ligado ao MCP tem-se o *transceiver* 82C51 da Philips, que simplesmente converte o sinal TTL vindo do controlador para uma sinalização diferencial, menos suscetível a interferências. Esse *transceiver* é totalmente compatível com o padrão ISO 11898 para sistemas de 24V, e foi desenvolvido para uso em sistemas veiculares, como ônibus e caminhões. Assim sendo, ele conta com recursos que agregam confiabilidade ao circuito, como controle de *slope* para redução de RFI (interferência de rádio-frequência), proteção contra curto-circuito, proteção térmica e alta imunidade a interferência eletromagnética (EMI). Ele transforma a sinalização TTL recebida do controlador CAN em sinalização diferencial, e vice-versa, de modo que o barramento se torna praticamente imune aos ruídos normalmente presentes em ambientes automotivos. Em sua ligação ao barramento CAN existe um resistor entre as linhas CANH e CANL para efetivar o descasamento de impedâncias.

Pelo esquema da figura 9 nota-se que não se utiliza o controlador de barramento CAN o MCP2510 quando se utiliza o PIC18F458. Isto é possível devido aos PIC´s da família 18F implementar em sua estrutura a interface CAN Bus. Com isso, torna-se desnecessário a utilização do chip MCP.

O MAX232 é usado para ligar a USART do PIC (Tecnologia TTL) em um computador do tipo PC (Padrão RS-232) pela interface serial deste último.

8.2 Programação Desenvolvida em Java

A programação desenvolvida em Java, mostrada a seguir, constituiu-se pelo desenvolvimento do programa Servidor que executa no WEB *Server* e que realiza a comunicação com os controladores de processo e com o Supervisório. É mostrado também o desenvolvimento do programa Supervisório que realiza a interface entre o usuário e os experimentos.

8.2.1 Programa Servidor

A seguir estão apresentadas as principais partes constituintes do programa Servidor.

8.2.1.1 Fluxo de Execução do programa Servidor

A figura 10 a seguir, exibe, esquematicamente, o fluxo de execução no tempo, das partes constituintes do programa Servidor após ser realizada a inicialização do mesmo.

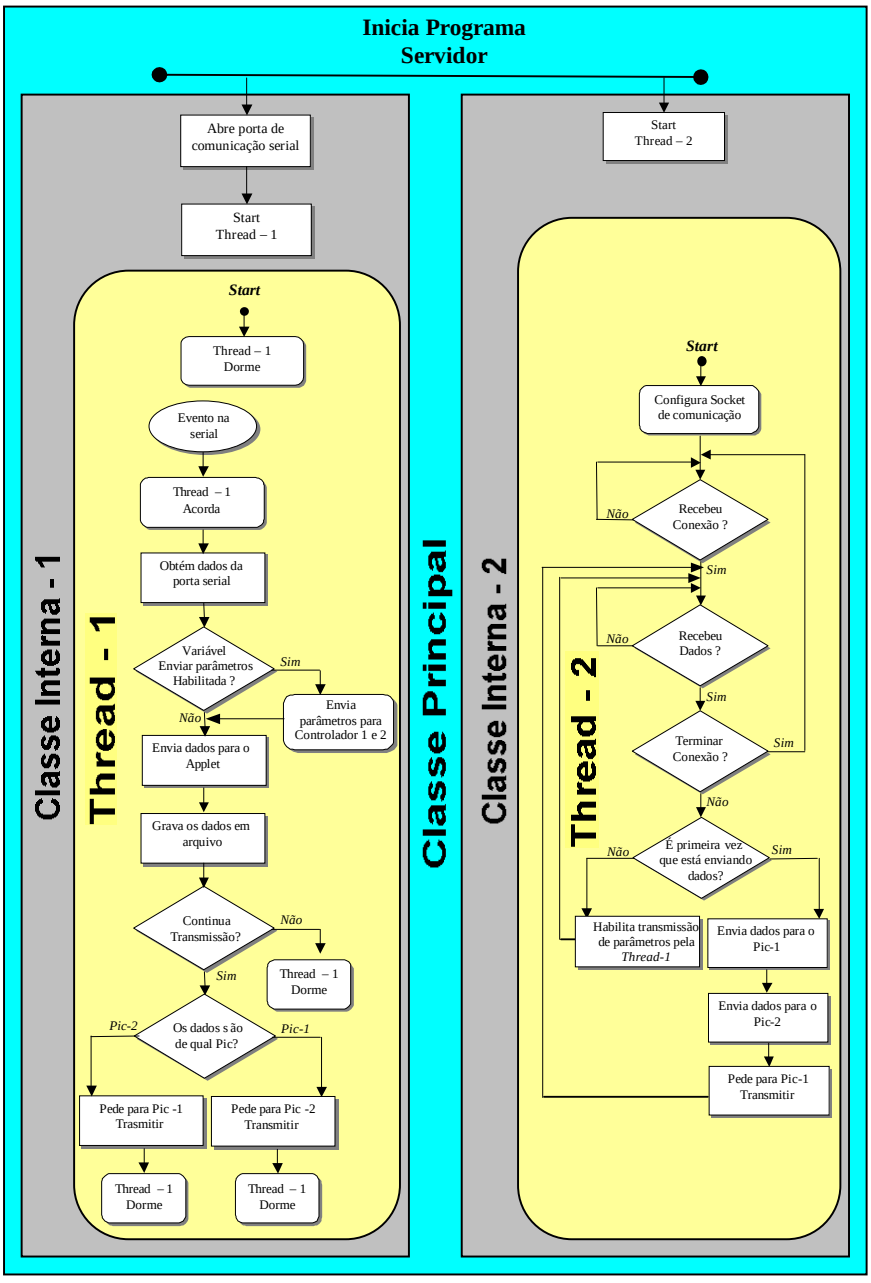

Figura 10: Diagrama de execução do programa Servidor

Como pode ser visto no diagrama anterior, assim que o programa Servidor (Classe Principal) entra em execução duas classes (Classes Internas), que possuem fluxos de execução (*Threads*) independentes, são instanciadas e inicializadas.

Uma das classes (Classe – 1) trata da comunicação do PC com o barramento RS-232, ou seja, realiza a comunicação bidirecional com os controladores de processo conectados ao barramento.

A outra classe (Classe – 2) trata da comunicação do PC com o usuário remoto (Supervisório). Esta, que também possui a característica bidirecional, permite que os dados do processo sejam enviados para o Supervisório continuamente e que, em qualquer momento, receba dados de configuração submetidos pelo usuário aos controladores de processo.

Esta abordagem permite que partes do programa que possuem fluxos de execução sejam executadas concorrentemente, ou seja, cada *thread* executa suas rotinas independentemente de outras. Vale ressaltar que Java é a única, entre as linguagens de programação de uso geral e popular, que torna as primitivas de concorrência disponíveis para o programador de aplicativos.

A divisão do programa Servidor em duas *threads* principais permitiu maior flexibilidade e eficiência na comunicação, já que o programa trata a comunicação com os microcontroladores e com o *Supervisório* de forma independente, dispensando, então, verificações desnecessárias pela chegada de novos dados que, por ventura, poderiam ter sido enviados pelo usuário no mesmo barramento. Dessa maneira, o atributo velocidade de execução das rotinas de recepção e envio de dados permitiu a consistência da característica *real-time* do sistema.

As duas subseções a seguir fazem um detalhamento das classes utilizadas na comunicação PC Servidor ←→ barramento e PC Servidor ←→ *Supervisório* respectivamente.

8.2.1.2 Comunicação PC Servidor ←→ Barramento

Como foi definida, na escolha da arquitetura do laboratório remoto, a comunicação do PC Servidor com o barramento segue as características do padrão serial RS-232C.

Para a utilização da porta serial do PC, Java possui uma biblioteca de classes que permite a programação, em alto nível, para configuração e utilização da mesma, sem ter que recorrer aos atributos de E/S do Sistema Operacional vigente.

Sempre que o programa principal, Servidor, é executado ele cria uma instância (Objeto) da Classe Interna – 2 (No código do programa esta classe possui o nome "Conexão"). Esta classe, logo em sua criação, faz uma varredura pelo sistema com o intuito de obter todas as portas conhecidas pelo SO. Dentre as portas verificadas o programa, primeiro procura pelas portas seriais e, então, verifica se entre elas existe a porta "COM2".

A escolha da porta "COM2" foi feita levando-se em consideração a sua presença na maioria dos PC's atuais. A não escolha da porta "COM1" se deve ao fato desta estar, quase sempre, associada ao *mouse* do computador.

Encontrada a porta "COM2" o programa tenta obter sua utilização para a comunicação com o barramento. Caso obtenha sucesso em sua abertura, o programa realiza a configuração dos parâmetros de transmissão que serão realizados na comunicação. Os parâmetros definidos foram os seguintes:

Velocidade de transmissão: 9600 bps
Número de bits de dados: 8 bits
Número de bits para o *Stop*: 1 bit
Bit de Paridade: False

Esta configuração dos parâmetros de transmissão é definida de forma idêntica nos terminais (controladores) que estão conectados ao barramento.

Após ajustar os parâmetros de transmissão são obtidos os *streams* de entrada e saída da porta. Estes permitem que o programa escreva e receba dados na porta como se estivesse realizando uma operação com arquivos.

Outra característica configurada no programa é o ouvinte de eventos na serial. Esta característica define que sempre que houver dado disponível na serial um evento é gerado no programa. Este atributo corresponde, com grande semelhança, às interrupções do SO. A escolha de habilitar este ouvinte de eventos é de grande importância no funcionamento desta classe já que sua *thread* estará sempre "dormindo".

Realizada toda a configuração da porta que fará a comunicação com o barramento, como foi definido anteriormente, o próximo passo do programa é criar sua *thread*. Assim que a *thread* é criada e inicializada a mesma é posta para "dormir". Este comportamento foi definido de modo que, a referida *thread,* não ocupe o processador desnecessariamente, ou seja, a mesma é executada apenas nos momento em que realmente necessite de sua execução.

A *thread* que encontra-se "dormindo" é "acordada" apenas quando ocorre evento na serial, sinalizando que chegou dado, ou quando algum dos métodos da classe à qual a *thread* pertence é chamado (empacota(); sendApp(); gravaArquivo()).

8.2.1.3 Protocolo Barramento → PC

O protocolo de comunicação entre os controladores de processo e o programa Servidor foi definido como a figura 11 abaixo:

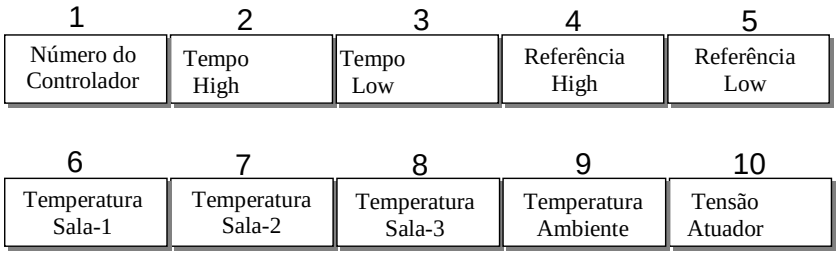

Figura 11: Sequência dos dados enviados pelos controladores ao programa Servidor

O primeiro campo transmitido (Identificador) é o dado que permite identificar de qual processo a mensagem pertence, ou seja, corresponde ao endereço da mensagem. Sendo este campo de 8 bits tem-se a possibilidade de obter 256 endereços distintos, ou seja, pode-se endereçar 256 entidades distintas no barramento.

O segundo e terceiro campo (Tempo) é o dado que informa o tempo desde o inicio do experimento. Considera-se o inicio do experimento o momento em que os primeiros parâmetros foram passados para o respectivo controlador e o mesmo iniciou a execução do experimento (programa interno).

O bit menos significativo da palavra tempo corresponde a 35 ms, sendo esta composta por 16 bits tem-se que o valor máximo de representação de tempo é de $35x10^{-}$

$^{3}x(2^{16}-1)$, ou seja, 2293 segundos. Vale ressaltar que o campo tempo não é composto por uma mensagem de 16 bits, mas sim por duas mensagens de 8 bits, já que a transmissão pela serial foi configurada como tendo 8 bits de dados.

O quarto e quinto campo (Referência) correspondem ao valor de temperatura em que deseja-se que a temperatura das salas estejam. O bit menos significativo da palavra referência corresponde a 0,5 ºC.

O sexto, sétimo e oitavo campo correspondem aos valores de temperatura das salas 1, 2 e 3 das respectivas salas controladas pelo microcontrolador. O bit menos significativo destas palavras corresponde a 0,5 ºC.

O nono campo (Temperatura Ambiente) corresponde à temperatura do ambiente em que se encontram os atuadores. O bit menos significativo corresponde a 0,5 ºC.

E por fim, o décimo campo (Atuador) corresponde à tensão média a qual estão submetidos os atuadores (secadores). Cada bit desta palavra corresponde a 1 Volt.

8.2.1.4 Execução ocorrida na chegada do protocolo "Barramento → PC"

Sempre que um dado chega na porta serial, a *thread-1*, que faz o tratamento da comunicação "barramento ←→ PC", encontra-se no estado "dormindo". Dessa maneira, como pode ser visto na figura 10, a primeira ação que ocorre é "acordar" esta *thread*. Estando a *thread-1* "acordada" a mesma obterá os dados, que se encontram disponíveis no buffer da porta, na seqüência definida pelo protocolo descrito anteriormente.

Assim que termina a etapa anterior os dados encontram-se armazenados em variáveis do programa. Com o intuito de transmitir um único bloco de dados para o Supervisório, é feito, então, um empacotamento das variáveis. Este empacotamento faz a conversão das variáveis do tipo "*short*" e "*byte*", que estão armazenando os dados, para o tipo "*String*". Desta maneira possibilita que as diferentes *strings* sejam concatenadas formando uma única *string* que contenha toda a informação do protocolo "Barramento → PC".

Após criar o pacote de dados o mesmo é enviado para o usuário remoto, que será recebido e tratado pelo Supervisório em execução.

O mesmo pacote de dados que é transmitido para o usuário é, também, registrado em arquivo. Este arquivo de dados estará disponível para o usuário remoto assim que o mesmo finalizar o experimento. Com este arquivo o usuário poderá fazer análises do experimento no modo *off-line* como, por exemplo, verificar as especificações de projeto do controlador através de curvas que poderão ser traçadas no *Matlab*.

Sabendo que a operação de Leitura/Escrita de um arquivo em disco é lenta, a operação de escrita do arquivo de dados em disco poderia se tornar um gargalo e prejudicar o comportamento *real-time* do sistema. Desta maneira, esta operação de escrita do arquivo de dados não é realizada em disco, mas sim em uma pasta mapeada na memória RAM da maquina (RAMDRIVE).

Caso o usuário remoto não tenha enviado dados para o programa Servidor indicando que é para parar o experimento, a *thread-1* continuará sua execução enviando mensagens para os controladores fazendo requisição de novos dados. Ou seja, pedirá para o microcontroladores transmitir mais dados (Ver figura 10).

Como no presente sistema possui apenas dois elementos conectados ao barramento, a política adotada para definir qual elemento irá usar o barramento para transmitir é feita alternando os pedidos de transmissão, ou seja, a *thread-1* verifica qual foi o último elemento que transmitiu os dados e então faz a requisição de novos dados para a outra entidade (Ver figura 10).

Assim que a *thread-1* faz a requisição de transmissão a uma dos elementos, a mesma termina a execução do evento que a “acordou” e então volta a “dormir”. Mas como a última operação que a mesma executou foi um pedido de transmissão aos controladores espera-se que ela “dorminá” por pouco tempo, pois logo, chegarão os novos dados e, assim, o ciclo se repetirá até que o usuário indique o término do experimento.

8.2.1.5 Protocolo PC → Barramento

Existem dois protocolos de comunicação entre o programa Servidor e os controladores de processo. Os mesmos estão representados nas figuras 12 e 13 abaixo:

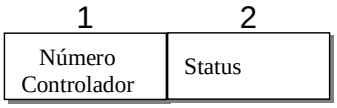

Figura 12: Protocolo PC → Barramento: Pedido de Transmissão

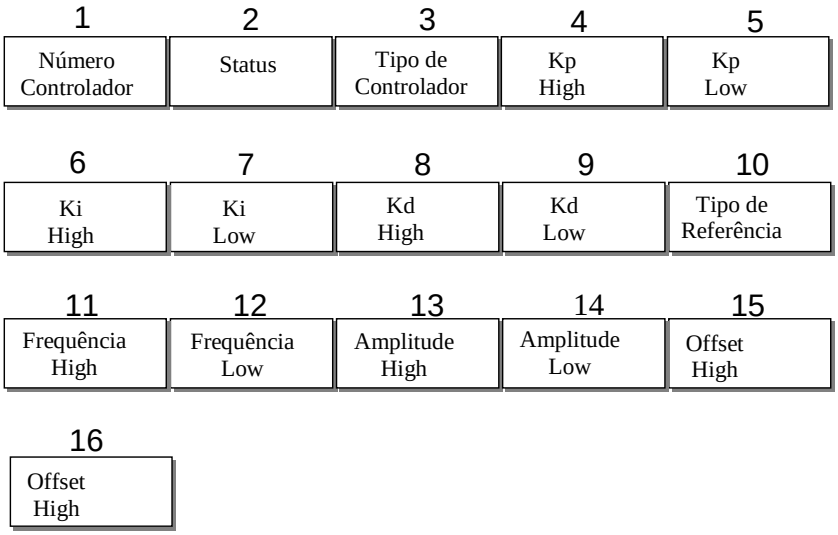

Figura 13: Protocolo PC → Barramento: Envio de Parâmetros

O primeiro, figura 12, corresponde ao pedido de transmissão por dados do processo que é feito pelo programa Servidor aos elemento que estão conectadas ao barramento.

O primeiro campo transmitido (Identificador) é o dado que permite identificar para qual elemento a mensagem está sendo transmitida, ou seja, corresponde ao destino da mensagem.

O segundo campo (Status), corresponde à identificação da mensagem, ou seja, permite ao elemento identificar que tipo de mensagem está chegando à ela. No caso de pedido de transmissão este campo possui o valor “00000001”.

O segundo protocolo, figura 13, é o que transmite os parâmetros que foram definidos pelo usuário para os controladores de processo conectados ao barramento.

O primeiro campo, como no caso do protocolo da figura 12, é o dado que permite identificar para qual controlador a mensagem está sendo transmitida.

O segundo campo (Status), corresponde à identificação da mensagem, ou seja, permite à entidade identificar que tipo de mensagem está chegando a ela. Diferentemente do protocolo da figura 12, este campo, no presente protocolo, tem a função de, também, informar o momento em que o controlador deve finalizar o controle do processo. A tabela - 03 a seguir mostra os diferentes valores para o campo "*Status"*, "Tipo Referência", "Tipo Controlador" e suas correspondências.

Status	**Mensagem**
00000001	Pedido de transmissão
00000010	Parâmetros sendo enviados
00000100	Finalizar Processo
Tipo Referência	**Mensagem**
00000001	Senoidal
00000010	Quadrada
00000100	Triangular
00001000	Degrau
Tipo Controlador	**Mensagem**
00000001	PID
00000010	*Fuzzy*

Tabela 3: Mensagens na comunicação PC → Controladores

8.2.1.6 Transmissão de dados para os controladores

Na primeira versão do programa Servidor, o qual foi projetado inicialmente, o tratamento dos dados que chegavam do Supervisório (*applet*) e sua transmissão para os controladores eram realizados pela *Thread-2.*

Assim que iniciaram os testes de comunicação percebemos que existia uma falha neste programa que culminava numa mistura dos dois protocolos discutidos anteriormente.

Como a *thread-1* fazia pedido de transmissão aos controladores e a *thread-2*, assim que chegavam os dados do *applet*, transmitia-os para os controladores houve, esporadicamente, a mistura dos dados devido ao seguinte fato: Quando uma das *threads* iniciava a transmissão do protocolo e o Sistema Operacional escalonava o processador para a outra *thread* que, por ventura, também iniciava a transmissão do outro protocolo ocorria em uma situação em que os controladores de processo não conseguiam identificar, obviamente, os dados que estavam chegando a eles.

Este problema foi solucionado utilizando o conceito de monitores. Java utiliza monitores para executar sincronizações. Todo objeto com métodos ***synchronized*** é um monitor. O monitor permite que uma *thread* de cada vez execute um método ***synchronized*** sobre o objeto. Isso é realizado bloqueando o objeto quando o método ***synchronized*** é invocado. Se houver vários métodos ***synchronized,*** somente um pode estar atuando em um objeto de cada vez; todos as outras *threads* tentando invocar métodos ***synchronized*** devem esperar.

Desta maneira, o método que enviava dados (sendProtocol()) e o que pedia transmissão (sendPic()) para os controladores foram definidos como métodos ***synchronized.*** Como estes métodos pertenciam a uma mesma classe (Classe Principal) e também ao mesmo objeto (Objeto único da classe principal) o conceito de monitores foi aplicado corretamente e, então, o problema foi resolvido com êxito.

No entanto, outros problemas apareceram. Devido à característica *full-duplex* da transmissão PC ←→ Barramento o programa Servidor foi projetado para atender a está característica, ou seja, mesmo quando os controladores estivessem transmitindo dados que seriam tratados pela *thread-1* o Servidor poderia transmitir parâmetros para os controladores através da *thread-2*. Porém foi identificado que quando os controladores iniciavam uma transmissão de dados para o PC e, antes que finalizassem a transmissão, chegassem dados do PC para os mesmos o restante dos dados que deveriam ser transmitidos para o PC não ocorria. Verificamos se havia falha no *software* dos controladores, porém não foi encontrado nada de errado. Acreditamos em falha de *hardware*, mas nada confirmado.

A solução encontrada para esta falha foi modificar a característica *full-duplex* para *half-duplex* através do programa Servidor. Nesta nova situação o Servidor não transmite parâmetros para os controladores enquanto os mesmos estão realizando a transmissão.

O esquema das duas *threads* foi mantido igualmente e com a mesma necessidade. A diferença é que neste novo sistema a *thread-2* não transmite os parâmetros do *applet* para os controladores, mas sim habilita uma variável que permite à *thread-1* transmitir os parâmetros do usuário (*applet*) para o controladores. Sempre que chega um dado na porta serial proveniente dos controladores a *thread-1* obtém os dados e, antes de fazer um novo pedido, verifica se a variável que permite a transmissão de parâmetros está habilitada. Caso verdadeiro a *thread-1* inicia a transmissão dos parâmetros para o barramento e, logo após, desabilita a variável e faz o pedido de transmissão de dados. Caso a variável de controle da transmissão de parâmetros não esteja habilitada a *thread-1* apenas faz o pedido de transmissão de dados aos controladores.

Vale ressaltar que, na primeira vez em que há o envio de parâmetros (iniciar o experimento) a transmissão tanto dos parâmetros como do pedido por dados é realizada pela *thread-2* já que não há possibilidade dos controladores estarem transmitindo e, então, gerando eventos na porta serial de modo que a *thread-1* acorde.

Destaca-se que esta nova configuração do programa Servidor não afetou a eficiência na comunicação, já que o programa continua tratando a comunicação com os controladores e com o *applet* de forma independente. Como a taxa de transmissão dos parâmetros é muito baixa, 16 bytes a cada mudança de configuração realizada pelo usuário, ter alterado o Servidor para não enviar os parâmetros enquanto os controladores estavam transmitindo não prejudicou a velocidade do sistema. Até porque a transmissão dos dados dos controladores para o PC não preenche completamente o tempo disponível para transmissão.

8.2.1.7 Comunicação PC Servidor ←→ Supervisório

Java implementa as comunicações baseadas em soquetes, que permitem aos aplicativos visualizar a rede como se fossem acessos E/S com arquivos, um programa pode ler de um soquete ou gravar em um soquete como se estivesse lendo ou gravando em um arquivo.

Com soquetes de fluxo, um processo estabelece uma conexão com outro processo. Enquanto a conexão estiver no ar, os dados fluem entre os processos em fluxos contínuos. Diz-se que os soquetes de fluxo fornecem um serviço orientado para conexão. O protocolo utilizado para transmissão é o popular *TCP* (*Transmission Control Protocol*) [7].

O primeiro passo no desenvolvimento de um aplicativo Servidor é criar um objeto ***ServerSocket*** com uma chamada ao construtor ***ServerSocket*** como abaixo:

```
ServerSocket server = new ServerSocket(porta, num);
```

Onde 'porta' especifica um número inteiro de porta disponível em que o Servidor espera conexões de clientes, e 'num' representa o número máximo de clientes que podem solicitar conexões ao Servidor. A configuração do programa desenvolvido tem como número de porta '3000' e número de conexões igual a '1'.

Cada conexão de cliente é gerenciada com um objeto *Socket*. Uma vez que o ServerSocket foi estabelecido, o Servidor ouve indefinidamente (ou bloqueia) uma tentativa de se conectar por parte de um cliente. Isso é realizado com uma chamada ao método accept de ServerSocket, como em:

```
Socket connection = server.accept();
```

que retorna um objeto ***Socket*** quando uma conexão é estabelecida. As figuras 14 e 15 a seguir mostram o Programa Servidor esperando e realizando uma conexão respectivamente.

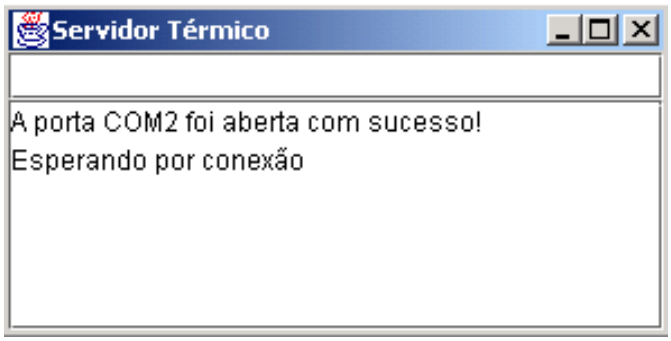

Figura 14: Servidor esperando por uma conexão.

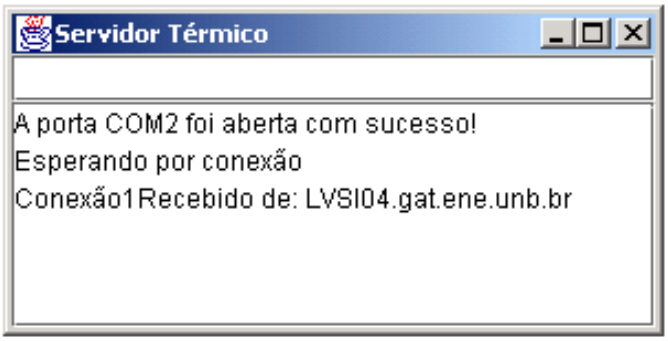

Figura 15: Servidor após receber uma conexão.

8.2.1.8 Estrutura dos parâmetros na comunicação Supervisório → PC Servidor

A interface do Supervisório (a*pplet*) permite que o usuário modifique, em tempo de execução do experimento, parâmetros relevantes ao processo. Dados relacionados com o tipo de controlador a ser empregado no sistema e referência de nível para os reservatórios podem ser atualizados e submetidos ao processo a qualquer momento pelo usuário.

Desse modo, assim que os parâmetros chegam na porta de comunicação com a rede (internet) a *thread-2* obtém estes dados e os colocam em variáveis do programa. A seqüência dos dados corresponde aos seguintes parâmetros:

1	2	3	4	5
Status	Número do Controlador	Tipo de Referência	Frequência	Amplitude

6	7	8	9	10
Offset	Tipo de Controlador	Kp	Ki	Kd

Figura 16: Sequência dos parâmetros enviados pelo *Supervisório* ao programa Servidor

Estes parâmetros chegam à porta como uma única mensagem que está contida em uma *string* de caracteres.

Obtida a *string* o programa Servidor utiliza uma classe chamada ***StringTokenizer*** que divide a frase em palavras individuais. Ou seja, ela obtém a *string* recebida do *Supervisório* e então a divide em segmentos de *strings* cada qual correspondendo aos parâmetros indicados na figura 16. A separação dos parâmetros é possivel ao programa

Servidor devido à construção da mensagem realizada pelo *Supervisório* que separa as diferentes informações por espaços em branco.

8.2.1.9 Estrutura dos parâmetros na comunicação PC Servidor → Supervisório

Assim que os dados chegam dos controladores eles são armazenados em variáveis do programa. Com o intuito de transmitir um único bloco de dados para o *applet*, é feito, então, um empacotamento das variáveis. Este empacotamento faz a conversão das variáveis do tipo "*short*" e "*byte*", que estão armazenando os dados, para o tipo "*String*". Desta maneira possibilita que as diferentes *strings* sejam concatenadas formando uma única *string* que contenha toda a informação que será enviada para o Supervisório. A seqüência dos dados montada no pacote segue a mesma indicada na figura 17 abaixo:

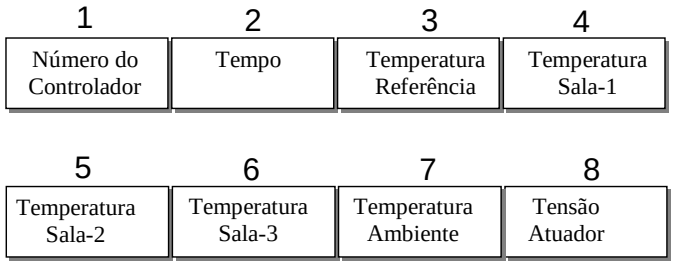

Figura 17: Sequência dos dados enviados pelo programa Servidor ao *Supervisório*

Vale ressaltar que a variável tempo, antes de ser convertida para o tipo *string*, é multiplicada por $35x10^{-3}$, já que o bit menos significativo do campo tempo vale 35ms.

8.2.2 Estabelecendo o Supervisório (*Applet* Java)

A seguir estão apresentadas as principais partes constituintes do Supervisório.

8.2.2.1 Fluxo de Execução do Supervisório

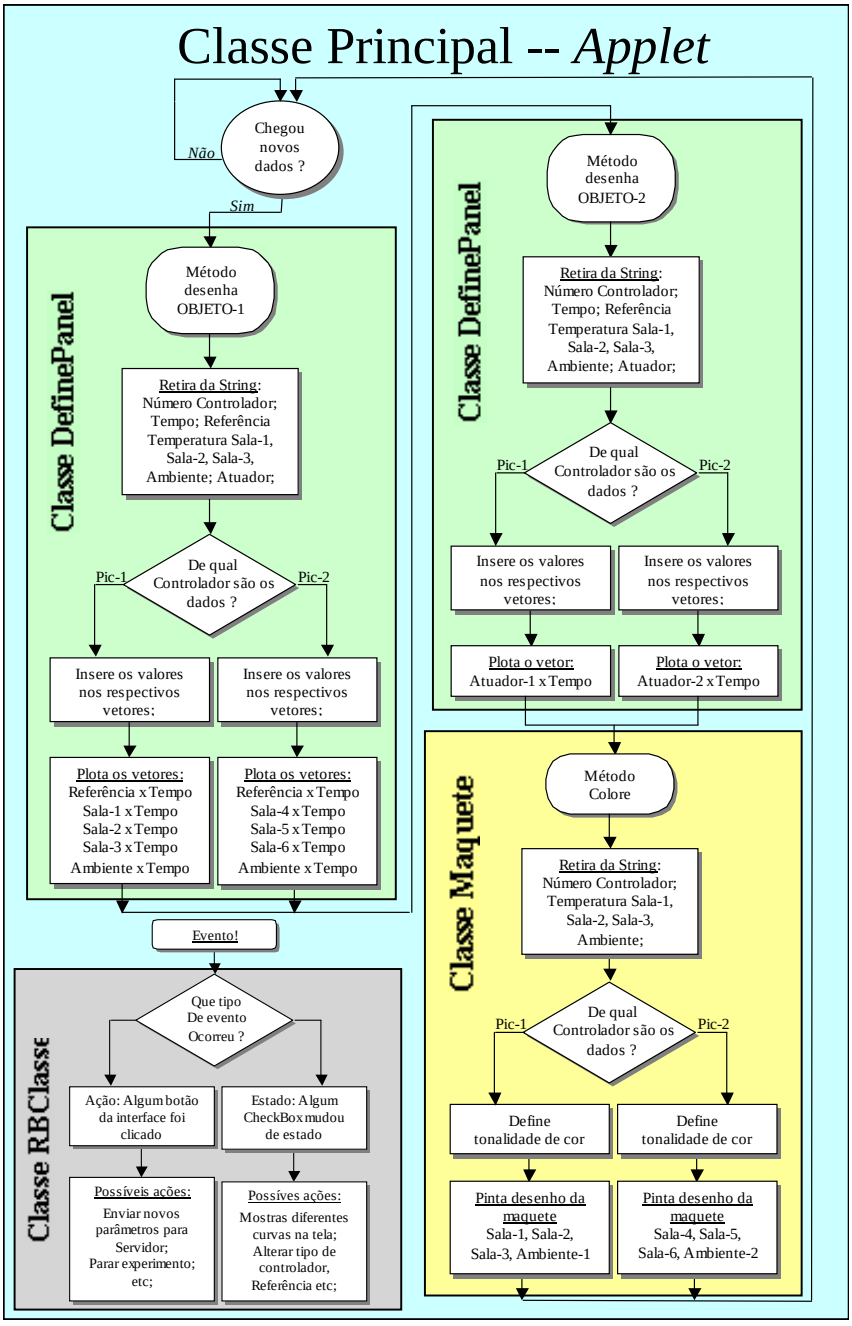

Figura 18: Diagrama do software Supervisório

Após enviar os parâmetros para o programa Servidor o Supervisório passa para o estado de espera de dados do Servidor, ou seja, do próprio processo em execução. O esquema mostrado na figura 18 representa este novo estado do Supervisório.

O esquema anterior exibe a seqüência dos vários módulos executados pelo Supervisório após o mesmo enviar os parâmetros do processo para o programa Servidor. Como pode ser visto, o programa fica em um *loop* até que um dado do Servidor chegue pela conexão realizada entre os dois *softwares*. Assim que um dado é obtido o mesmo é passado como parâmetro do método “desenha”.

O método “desenha” inicialmente faz o desmembramento das informações contidas no pacote de dados. Realizada esta operação, os dados que, originalmente, são do tipo ASCII são convertidos para ponto flutuante.

Como pode ser visto no esquema, existem dois objetos de uma mesma classe que possuem o método “desenha”. Isto foi feito com o intuito de obter duas janelas, independentes, para o desenho das curvas “temperatura *x* tempo” e das curvas “tensão no atuador x tempo”.

O método “desenha” é implementado para executar operações distintas sendo a operação desejada especificada na criação do objeto (Instanciação). A diferença entre as operações está nos vetores a serem plotados.

Após a retirada dos valores do pacote de dados e a conversão para ponto flutuante os mesmos são inseridos nos respectivos vetores. Ressalta-se que novos dados correspondem a novos elementos no vetor. Assim que os novos elementos são criados é feita uma chamada de método, do ambiente Java, que plota na tela os respectivos vetores.

Vale destacar que a chamada do método “colore” da classe maquete é realizada após ser feito todo o procedimento de plotagem do objeto-1 e do objeto-2, e que os parâmetros passados para esta classe são os mesmos que foram passados para o objeto-1 e objeto-2, ou seja, o mesmo pacote de dados oriundos do programa Servidor.

8.2.2.2 Interface gráfica do Supervisório

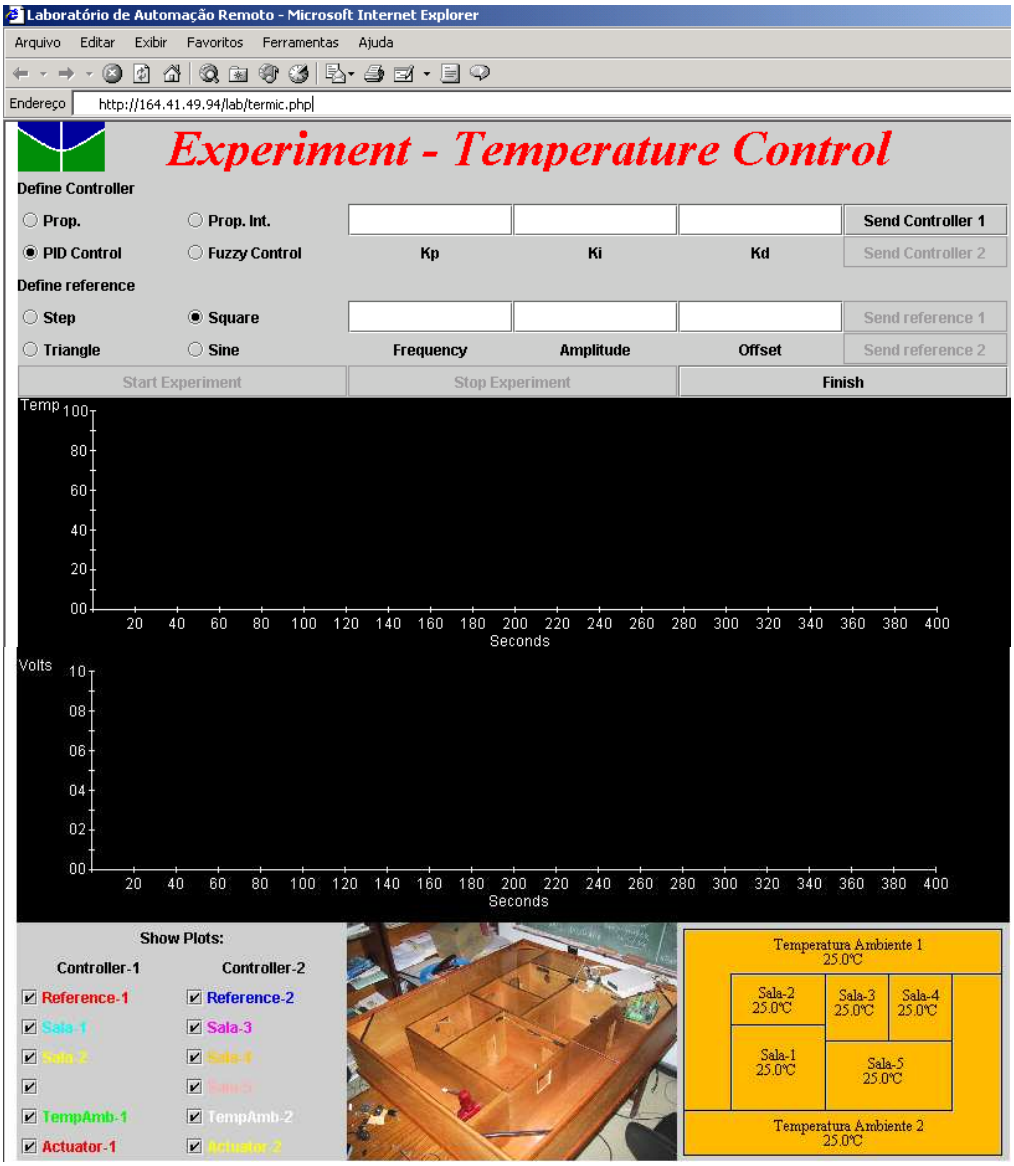

Figura 19: Interface gráfica do Supervisório

Após a conexão com o programa Servidor, o Supervisório passa para o estado de espera de dados a serem inseridos pelo usuário. Como pode ser visto na figura 19 acima, o usuário tem a possibilidade de definir o tipo de controlador que será submetido, o experimento bem como a referência de temperatura para as salas da maquete.

Assim que o usuário define os parâmetros para o experimento os mesmos devem ser enviados para o Servidor clicando no botão enviar Controlador/Referência. Porém, antes de serem submetidos, os parâmetros são passados por uma inspeção no intuito de verificar a validade dos mesmos. Esta crítica de dados é feita no próprio *Supervisório*, pois assim, evita-se tráfego na rede de parâmetros que não fazem sentido ao experimento. As figuras 20 e 21 a seguir mostram alguns dos erros que podem ocorrer.

- Campo digitado com caracteres não numéricos:

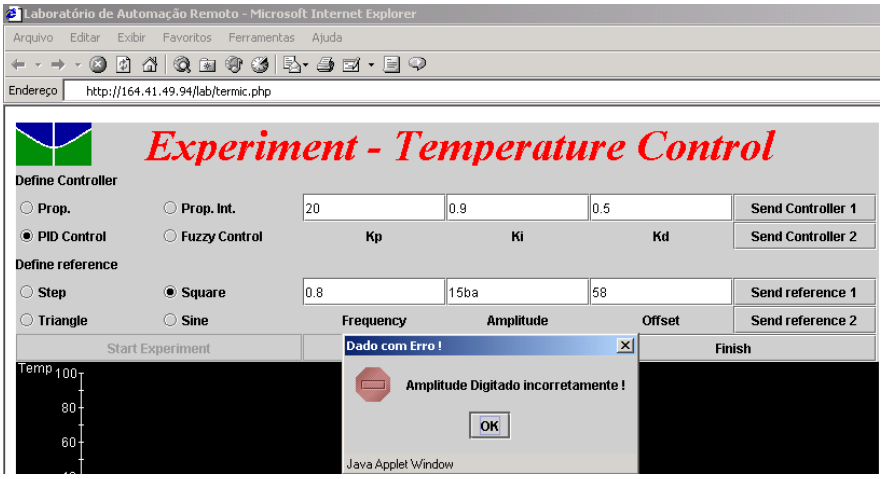

Figura 20: Critica de dados realizada pelo Supervisório: Campo digitado incorretamente

- Campo em branco:

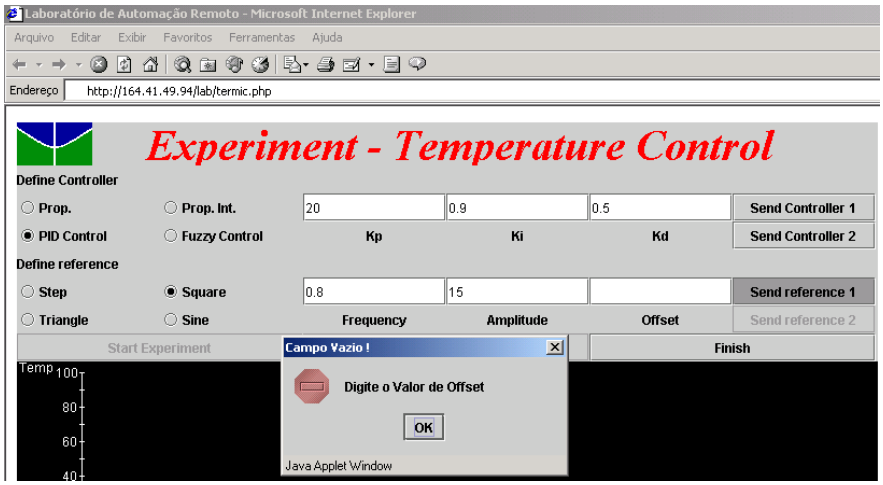

Figura 21: Crítica de dados feita pelo Supervisório: Campo em Branco

A interface gráfica do Supervisório possibilita ao usuário visualizar as diferentes curvas, referentes às temperaturas das diferentes salas, separadamente em uma tela, como também, a visualização de quaisquer curvas simultaneamente na mesma tela. Esta opção contribui para um melhor entendimento de cada sinal e também como uma ferramenta de comparação entre as respostas dos controladores. Esta mesma opção é disponibilizada para a tela dos atuadores. As figuras 22 e 23 abaixo mostram está característica do Supervisório.

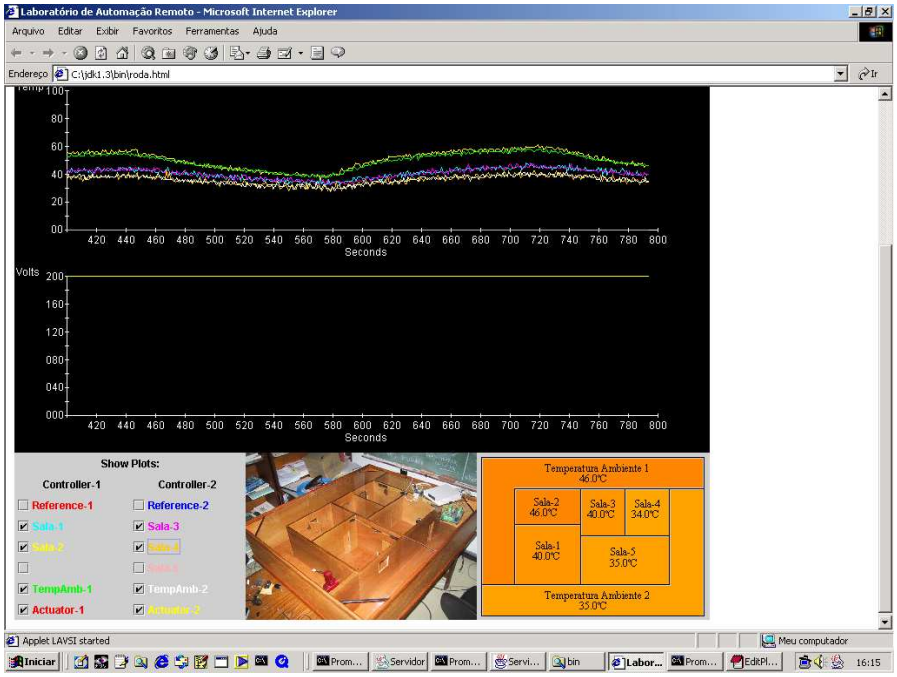

Figura 22: Tela com todas curvas plotadas.

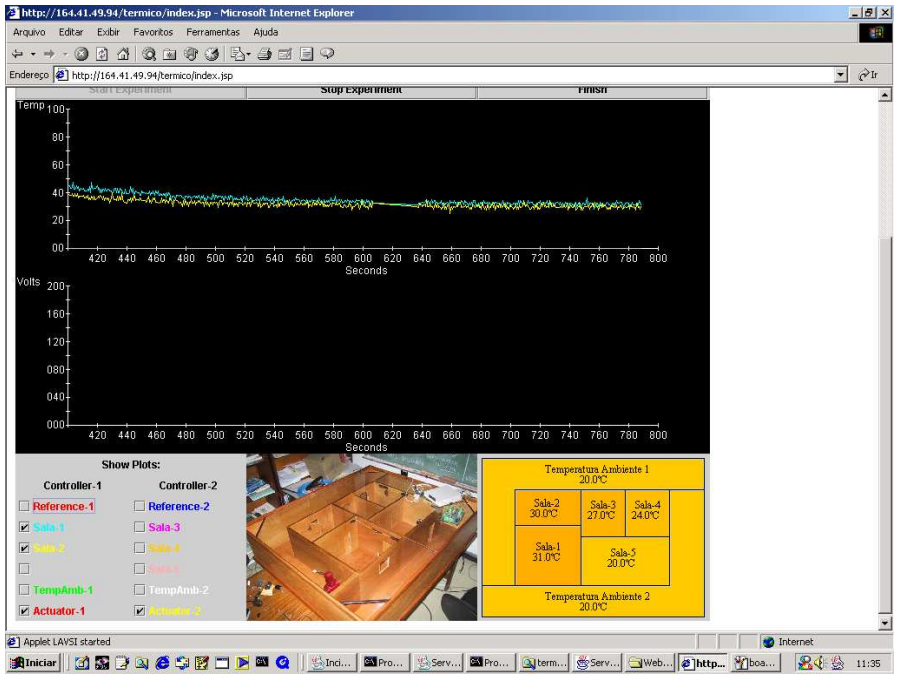

Figura 23: Tela com apenas a duas curvas sendo plotadas.

Como pode ser visto nas duas figuras anteriores, o usuário tem a possibilidade, também, de acompanhar a temperatura nas diferentes salas da maquete através da variação da tonalidade das cores nas respectivas salas.

Quando o botão "*Stop Experiment*" é clicado, o Supervisório transmite a informação para o programa Servidor que, por sua vez, transmite a informação para os controladores de processo indicando que o experimento deve ser parado. A partir de então, a conexão entre os dois programas é finalizada.

8.2.3 Disponibilidade dos dados do experimento

Assim que o usuário termina o experimento, clicando no botão "*Exit Experiment*", uma nova janela (HTML) é aberta no *browser* do usuário. Esta nova página possui um *link* para o arquivo de dados gerado durante todo o tempo em que o processo esteve em execução, ou seja, o arquivo que foi gravado pelo programa Servidor na unidade RamDrive.

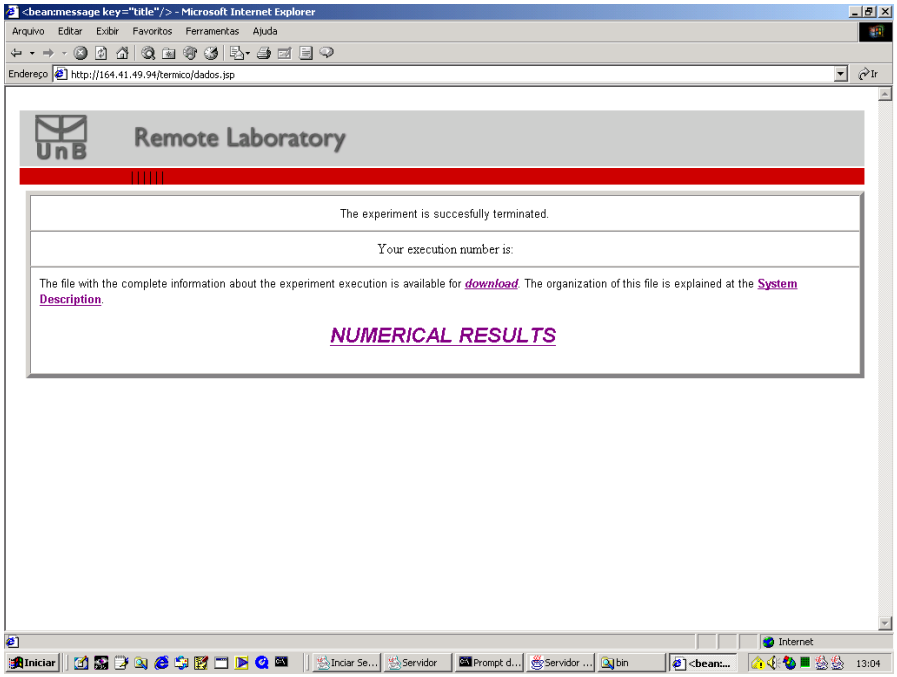

Figura 24: Interface após sair do experimento

A figura 25, a seguir, mostra uma tela de dados gerados em um experimento obtida pelo usuário após o encerramento do experimento.

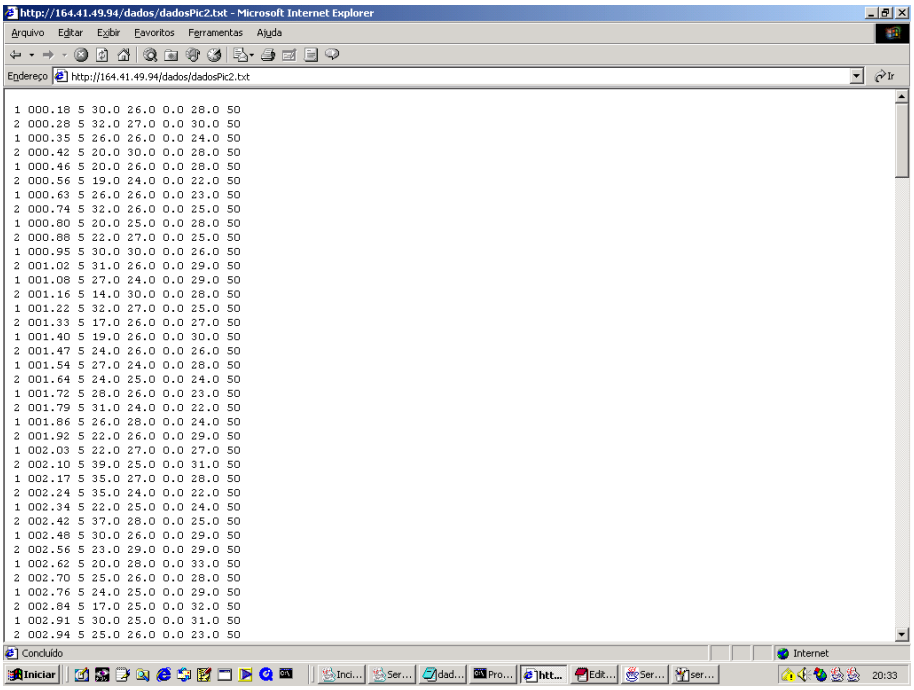

Figura 25: Dados gerados em um experimento.

8.3 Programação Desenvolvida em Assembly

8.3.1 Comunicação Serial

Primeiramente foi feito um programa para realizar testes de comunicação serial entre o computador e o microcontrolador. Neste programa faz-se uso de um módulo interno do PIC destinado à comunicação serial. Este módulo é a USART (*Universal Syncrhronous and Asynchronous Receiverand Transmitter*). A USART é um protocolo universal que possui dois modos de funcionamento: o sincronizado e o não sincronizado.

O modo utilizado no projeto foi o modo assíncrono, visto que o usuário do sistema pode entrar com valores quando ele sentir necessidade, fazendo com que se tenha a entrada e saída de valores no sistema, algo bastante aleatório.

A comunicação é feita somente com duas vias, entretanto, como este modo não é sincronizado, estas duas vias são utilizadas para dados. Uma delas para transmissão (TX) e a outra para recepção (RX). Isto possibilita que as informações sejam enviadas e recebidas ao mesmo tempo, cada qual na sua via. Este recurso é conhecido com *Full-Duplex*. Este é o mesmo modo utilizado na porta serial dos computadores.

O programa desenvolvido em *Assembly* para teste da comunicação serial pode ser verificado no ‘Apêndice A’. Assim como este programa todos os outros possuem uma espécie de cabeçalho de programação que possibilita o microcontrolador estabilizar suas funções internas permitindo o funcionamento normal do sistema. O fluxograma deste cabeçalho mencionado está representado na figura 26.

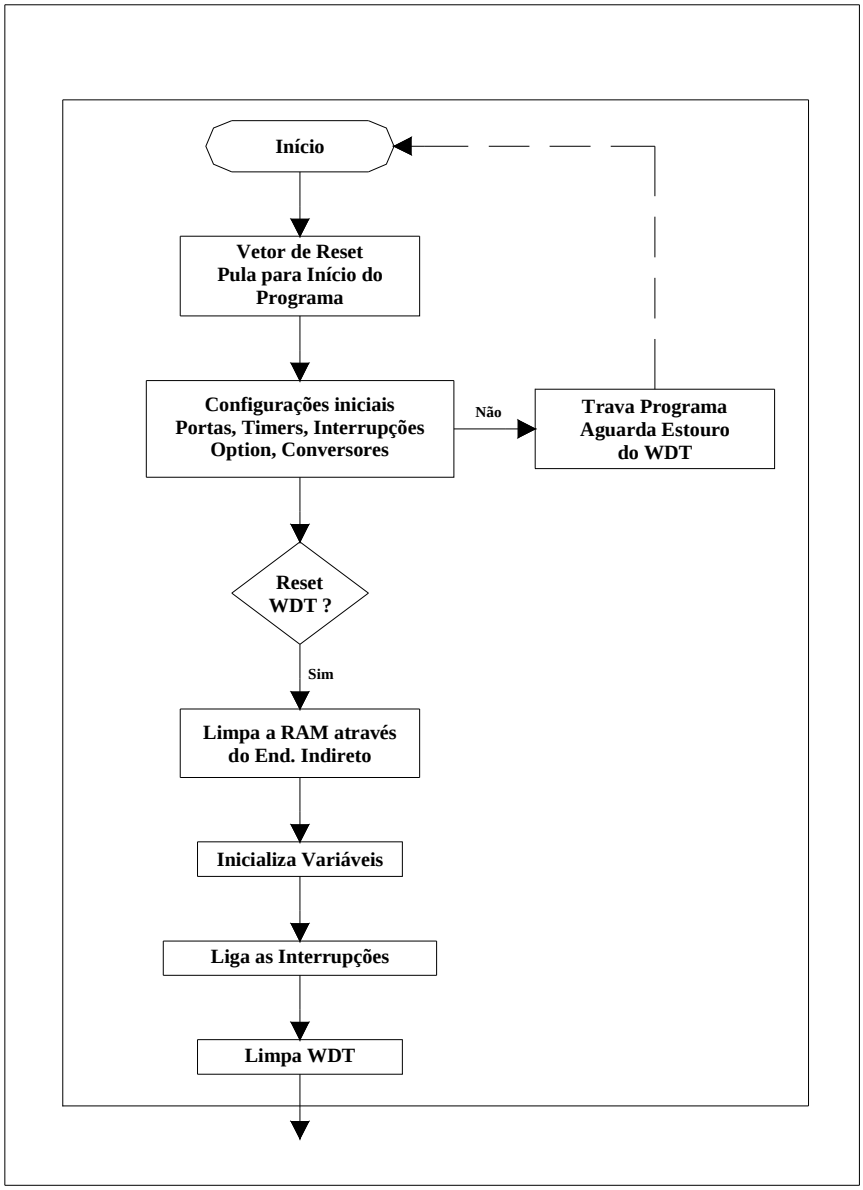

Figura 26: Fluxograma do Cabeçalho Geral

A funcionalidade de cada bloco do fluxograma apresentado acima está explicada no ‘Apêndice A’.

8.3.2 Módulo do Protocolo

Neste programa é utilizada a comunicação USART, anteriormente mencionada, e tem-se a seguinte configuração:

PIC (recebe) - PC (transmite)	PC (recebe) - PIC (transmite)
Pic	NUMPIC
Status	TEMPO_HI
Tipo_controle	TEMPO_LO
Kp_hi	REF_HI
Kp_lo	REF_LO
Ki_hi	TEMP_SALA1
Ki_lo	TEMP_SALA2
Kd_hi	TEMP_SALA3
Kd_lo	TEMP_SALA4
Onda_ref	SECADOR_HI
T_sobre2_hi	SECADOR_LO
T_sobre2_lo	
Amp_hi	
Amp_lo	
Off_hi	
Off_lo	

Tabela 4: Variáveis de transmissão e recepção PIC ←→ PC

As variáveis da tabela acima são enviadas na seqüência apresentada (ex: pic, status, tipo_controle, off_lo) do PC para o PIC ou do PIC para o PC (ex: NUMPIC, TEMPO_HI, SECADOR_2). O detalhamento deste módulo é descrito no Apêndice D.

8.3.3 Módulo das Ondas

Antes de se deslocar para uma das quatro opções de ondas disponíveis no sistema supervisório, o programa pergunta para qual tipo de onda o usuário deseja ir (Senoide, Quadrada, Triangular ou Degrau) e de acordo com o valor passado do computador pelo usuário (tecla das opções de ondas na tela do supervisório) ao microcontrolador o programa desloca para uma subrotina determinada pelos valores binários atribuídos ao byte Onda_ref:

Onda_ref	Tipo de Onda
1 (bit 0 = 1)	Senoidal
2 (bit 1 = 1)	Quadrada
4 (bit 2 = 1)	Triangular
8 (bit 3 = 1)	Degrau

Tabela 5: Valores de Onda_ref e as respectivas ondas de referência

O deslocamento provocado pelo byte Onda_ref promove a chamada das subrotinas (módulos) dos tipos de ondas. As ondas Quadrada, Triangular e Degrau estão melhor detalhadas nos Apêndice E, F e G, respectivamente. A onda senóide não foi desenvolvida e, portanto não tem-se seu detalhamento.

8.3.4 Módulo do Controle PID

Iniciou-se o programa ressetando as variáveis de entrada do usuário remoto - amplitude, freqüência, *offset*, ganhos (kp, kd e ki) – e outras internas – temperaturas das salas, entradas de controle para alimentação dos secadores, referências (*setpoints* para as ondas), erro (Erro = Temperatura – Referência), erros anteriores e integral dos erros. Estas variáveis foram *ressetadas* antes de iniciar o laço do LOOP principal.

No inicio do módulo do Controle PID tem-se a conversão Analógica/Digital do valor coletado nos sensores de temperatura (Temperaturas da sala 1 e 5), e guardados em TEMP1 e TEMP5 para futuras alterações no controle PID.

Logo após a conversão segue os passos de atualização do processo de controle como:

Erro anterior = Erro

Erro = Referência – TEMP1 ou TEMP5

Integral do erro = Integral do erro + Erro

Para finalizar este módulo implementou-se a seguinte equação de controle PID:

U = Kp*Erro + (Ki*Integral)*Ts + (Kd*(Erro – Erro anterior))*Fs;

As variáveis Ts e Fs representam a taxa de amostragem do sistema utilizando o controle PID. O cálculo feito para a obtenção destes valores foi o seguinte:

PIC 1

Ts = N Conversões + LOOP do Controle PID + LOOP da Onda Desejada

3 Conversões = $X = 3 * (T_{Adequação\_do\_capacitor} + T_{Conversão} + T_{\text{Re}ligamento\_do\_capacitor})$

$X = 3 * (40 + 12 * T_{AD} + 2 * T_{AD})$, onde:

$T_{AD} = 32 / F_{OSC}$ e $F_{OSC} \rightarrow$ Freqüência do Oscilador Externo.

X = 498 μs

Onda Quadrada:

Ts = 498 μs + 1164 μs = 1662 μs
Fs = 602 Hz

Onda Quadrada:

Ts = 498 μs + 1114 μs = 1612 μs
Fs = 620 Hz

PIC 2

Ts = N Conversões + LOOP do Controle PID + LOOP da Onda Desejada

4 Conversões = $X = 4 * (T_{Adequação\_do\_capacitor} + T_{Conversão} + T_{\text{Re}ligamento\_do\_capacitor})$

$X = 4 * (40 + 12 * T_{AD} + 2 * T_{AD})$, onde:

$T_{AD} = 32 / F_{OSC}$ e $F_{OSC} \rightarrow$ Freqüência do Oscilador Externo.

X = 664 μs

Onda Quadrada:

Ts = 664 μs + 1164 μs = 1828 μs
Fs = 547 Hz

Onda Quadrada:

Ts = 498 μs + 1114 μs = 1778 μs
Fs = 562 Hz

A implementação do Controle PID é vista no fluxograma apresentado na figura 27 abaixo.

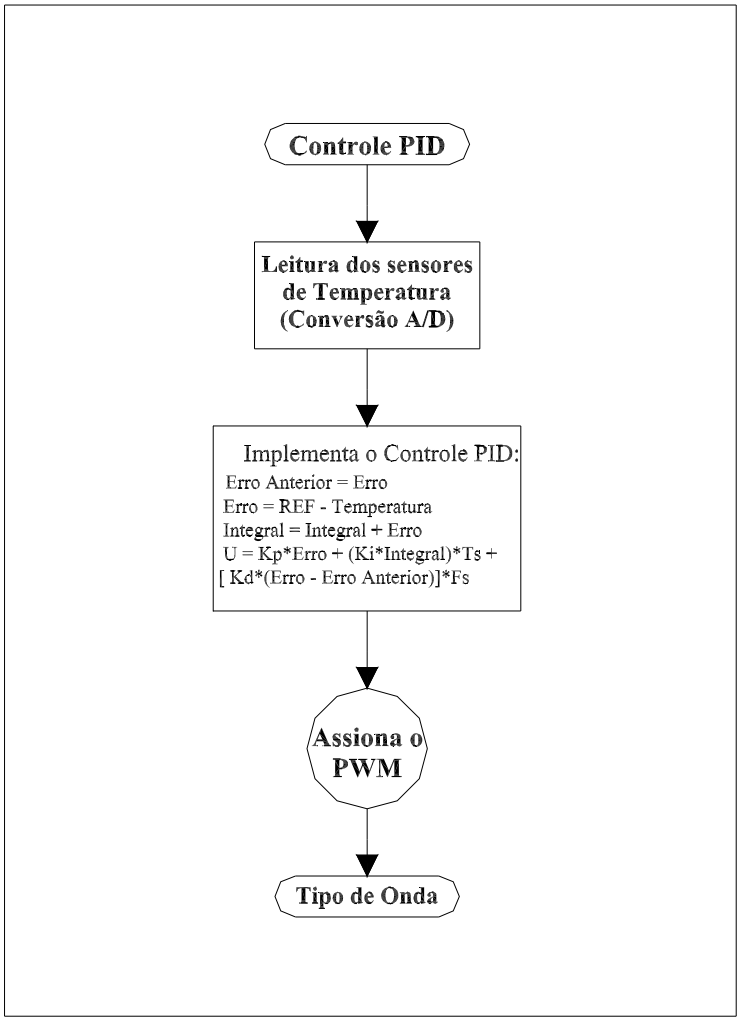

Figura 27: Fluxograma do módulo "Controle PID"

9 Conclusões

O funcionamento do laboratório remoto apresentou-se de maneira eficaz dentro do previsto para o projeto em questão. Do ponto de vista educacional, o laboratório remoto, no contexto de ensino à distância, apresenta-se como uma ferramenta no auxílio do ensino de engenharia de controle, pois há um nível elevado de interação entre usuário e experimento o que possibilita que conceitos básicos de controle clássico e inteligente sejam submetidos e verificados em experimentos reais resultando em uma maior compreensão por parte do usuário.

A escolha da arquitetura do laboratório apresentou-se bastante eficiente. As exigências básicas para a disponibilização de um laboratório remoto para controle de processos foram alcançadas. A característica *real-time* foi atingida com sucesso, onde foi garantido o controle eficiente do processo térmico como também a constante interação do usuário com o experimento.

O uso do barramento RS-232 multi-ponto apresentou-se como uma ótima solução para aplicações em que não se exige alta velocidade de transmissão. Como realizado no projeto, ou seja, duas entidades conectadas ao barramento, a funcionalidade da transmissão executou-se perfeitamente, já que não foram observados erros na transmissão mesmo quando a transmissão era realizada exaustivamente pelas duas entidades.

A banda de transmissão disponibilizada pelo barramento foi mais do que suficiente para a interação e o monitoramento por parte do supervisório com as duas entidades. A expressão “mais do que suficiente” está sendo utilizada já que as entidades não utilizavam todo o tempo disponibilizado para transmissão pelo barramento.

Devido à característica da interface do supervisório que não plotava na tela variações de tempo menores que 0,58s (Cada pixel na tela corresponde a 0,58s) a

transmissão de informações com intervalo menor do 0,58s seria desnecessária. Dessa maneira, sempre que realizava-se o pedido de transmissão de dados aos controladores estes aguardavam 0,25s antes de transmitirem dados.

Como o pedido de transmissão era intercalado entre os controladores verificou-se, através do tempo entre dois blocos de dados provenientes de um mesmo controlador, que o tempo de processamento do PC e dos PIC´s mais o tempo de transmissão dos dados estavam em torno de 0,06s. Pois o tempo entre dois blocos de dados de um mesmo controlador era de aproximadamente 0,56s e sabendo que cada controlador aguardava 0,25s para transmitir tem-se: 0,56 – (0,25+0,25) = 0,06. Conclui-se então que o barramento está sendo utilizado apenas 11 % do disponível (0,06/0,56).

Vale destacar que esta porcentagem de utilização do barramento pode, ainda, ser diminuída aumentando a taxa de transmissão dos controladores. A atual taxa de 9600 bps pode ser aumentada para algo em torno de 19200 bps.

O microcontrolador adotado, o PIC16F877A apresentou memória de dados suficiente tanto na implementação do módulo do controle proporcional-integral-derivativo, quanto na implementação do controlador *fuzzy*. Um problema em princípio encontrado foi à impossibilidade de programação em mais alto nível com o uso da linguagem C. É válido ressaltar que a partir dos PIC's da família 17F e 18F já existem ferramentas do próprio MPLAB que possibilitam a edição e compilação de programas desenvolvidos na linguagem C. Por outro lado, o uso da programação em *Assembly* proporcionou a confecção de programas mais enxutos do que em linguagens de mais alto nível, possibilitando a gravação dos programas desenvolvidos (Controle Clássico e Controle *Fuzzy*) por completo em cada PIC sem ter problemas de memória de dados. Problemas estes encontrados pelo ex-aluno da UnB, Fernando de Melo Luna Filho, em seu projeto de graduação devido o uso de linguagem de mais alto nível.

Além de boa quantidade de memória, o microcontrolador mostrou outras inúmeras vantagens para o controle de processos térmicos e para a manutenção do sistema supervisório como: três contadores (*timers*), dois módulos que servem para captura, comparação e geração de sinal PWM, conversor analógico-digital embutido, uso de até 33 pinos para operações de entrada e saída, módulo interno destinado à comunicação serial no modo síncrono e assíncrono (barramento RS-232C).

A velocidade de operação do microcontrolador, 0,222μs/instrução, possibilitou uma alta taxa de amostragem, permitindo implementar os algoritmos de controle na forma "quasi-contínua".

A maquete usada como sistema para implementação dos algoritmos de controle e estudo comparativo do desempenho dos mesmos foi bastante útil. Algumas não-linearidades que não foram consideradas na modelagem dos processos, como a interação com o meio ambiente e a transferência de calor envolvida pelas paredes da maquete representantes das divisórias dentro do escritório, não apresentaram influência que comprometesse os resultados alcançados. Esta maquete poderia ser usada em projetos futuros envolvendo controle de temperatura em ambientes.

Os elementos utilizados para controle dos processos dentro da maquete estudada poderiam ser usados para controle de temperatura usando como acionamento o aparelho de ar-condicionado. Alterações substanciais seriam necessárias no projeto do *driver* de potência, já que este aparelho possui potência muito maior do que a potência do secador de cabelo usado com "1kW".

Feitas estas modificações, o sistema proposto seria implementado e testado em laboratório, consistindo em um primeiro passo em direção à transformação deste projeto em um produto disponibilizado no mercado com custo baixo, resultando em uma alta taxa de retorno para os usuários que investissem no mesmo. A programação e operação seriam simples, tendo a maioria das tarefas automatizada e exigindo dos operadores apenas a definição das regras lingüísticas.

10 Bibliografia

[1] **Catálogo de busca na *internet*:** http://www.google.com/

[2] **CAN in Automation**: http://www.can-cia.de/

[3] **"Matéria da Capa", Oficina Brasil:** http://www.oficinabrasil.com.br/edicoes/Mai_2002/tecnico.htm

[4] **Aplicon Liveline Industrial IT and Instrumentation:** http://www.amplicon.co.uk/can.html

[5] **Synergetic Micro Systems**: http://www.synergetic.com/

[6] David José de Souza: **Desbravando o PIC – Baseado no Microcontrolador PIC16F84**. Editora Érica 2º Edição - 2000.

[7] **Microchip Web Site**: http://www.microchip.com/1000/index.htm

[8] Malcon A. T., Claudio L., Sérgio S.: **Comunicação de dados usando Linguagem "C**". Editora da FURB, São Paulo – 1996;

[9] **The Source for Java™ Technologyn** http://java.sun.com/products/javacomm/javadocs/Package-javax.comm.html

[10] **Barramento RS-485**: http://www.iee.efei.br/~gaii/rs485/hp_rs485.htm

[11] **Barramento CAN:** http://cliente.escelsanet.com.br/fmotoki/docs/mono.pdf

[12] **Zelenovsky, Ricardo** – conversa pessoal em maio de 2003;

[13] Ian S. Shaw, Marcelo G. Simões: **Controle e Modelagem *Fuzzy*.** Editora Edgard Blücher, São Paulo – 1999;

[14] David José de Souza; Nicolas C. Lavinia: **Conectando o PIC – Explorando Recursos Avançados.** Editora J. J. Carol 1ª Edição, São Paulo – 2002;

[15] **Data-sheet dos seguintes componentes**: PIC16F877A, LM35DZ, BC548, MAX232, D1N4004;

[16] **Empresa Mosaico Engenharia:** www.mosaico-eng.com.br

[17] **Materiais didáticos da linguagem Java:** www.cic.unb.br/docentes/fernando/matdidatico/intro.htm

[18] Apostila "**Mini-Curso de Microcontrolador**" feita por José Edson dos Santos Marinho e Edrialdo do Santos Marinho (Revista Saber Eletrônica);

[19] Apostilas "**A linguagem Java**", "**Programação Distribuída Usando Java**", "**Programação Cliente/Servidor Usando Java**", "**A Linguagem JavaScript**" feitas por Fernando Albuquerque (professor de Ciência da Computação da UnB);

[20] Apostila "**Mini-Curso de Sistemas Inteligentes – Redes Neurais e Lógica Fuzzy**" feita por Adolfo Buschspiess (professor de Engenharia Elétrica da UnB);

[21] Revista **Mecatrônica Atual – nº 4.** Editora Saber Ltda. São Paulo – Junho/2002.

[22] Antônio A. C. Leite, Ênio S. Leite: Trabalho de Graduação 2 "**Controle de Processo Térmico Multivariável visando a Racionalização de Energia**". Universidade de Brasília - 2002

[23] Fernando de Melo L. Filho: Trabalho de Graduação 2 "**Controle Inteligente para Automação Predial**"**.** Universidade de Brasília - 2003

Apêndice A: Programa de Comunicação Serial

Este apêndice trata da implementação do programa da comunicação serial no compilador MPASM. O MPASM é umas das ferramentas de compilação do *software* MPLAB. Abaixo segue o programa feito em *Assembly*.

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *   Programa feito para testar a comunicação entre a porta serial do     *
; * computador e do microcontrolador PIC16F877A.                          *
;*************************************************************
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *   VERSÃO : 1.0                                                        *
: *   DESENVOLVEDORES:                                                    *
: * Alexandre Silva Souza                                                 *
; * Leandro Matos de Almeida Ramos                                        *
; *   PROGRAMA:                                                           *
; *  serial.asm                                                           *
; *   DATA : 10/03/2003                                                    *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                    CONFIGURAÇÕES PARA GRAVAÇÃO                        *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

 __CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &
_PWRTE_ON & _WDT_ON & _XT_OSC

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *              DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC                  *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE
;  OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE
;  DE REDIGITAÇÃO.

        list                p=16f877A               ; MICROCONTROLADOR UTILIZADO
        #include"E:\SERIAL\p16f877A.inc"        ; USO DA BIBLIOTECA p16f877A.inc

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                  DEFINIÇÃO DOS BANCOS DE RAM                          *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR
;  ENTRE OS BANCOS DE MEMÓRIA.

        #DEFINE BANK1 BSF STATUS,RP0                ; SELECIONA BANK1 DA MEMORIA RAM
        #DEFINE BANK0 BCF STATUS,RP0                ; SELECIONA BANK0 DA MEMORIA RAM
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                  DEFINIÇÃO DAS VARIÁVEIS                    *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO NO FINAL DO BANCO 0, A PARTIR
;  DO ENDEREÇO 0X70, POIS ESTA LOCALIZAÇÃO É ACESSADA DE QUALQUER BANCO,
;  FACILITANDO A OPERAÇÃO COM AS VARIÁVEIS AQUI LOCALIZADAS.

        CBLOCK      0X70            ; POSIÇÃO COMUM A TODOS OS BANCOS

                W_TEMP              ; REGISTRADOR TEMPORÁRIO PARA W
                STATUS_TEMP         ; REGISTRADOR TEMPORÁRIO PARA STATUS
                PCLATH_TEMP         ; REGISTRADOR TEMPORÁRIO PARA PCLATH

        ENDC

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *            VETOR DE RESET DO MICROCONTROLADOR      *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

        ORG     0X0000              ; ENDEREÇO DO VETOR DE RESET
        GOTO    CONFIG              ; PULA PARA CONFIG DEVIDO A REGIÃO
                                    ; DESTINADA ÀS INTERRUPÇÕES

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *******
; *          VETOR DE INTERRUPÇÃO DO MICROCONTROLADOR      *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *******
;  POSIÇÃO DE DESVIO DO PROGRAMA QUANDO UMA INTERRUPÇÃO ACONTECE

        ORG     0X0004              ; ENDEREÇO DO VETOR DE INTERRUPÇÃO

;  É MUITO IMPORTANTE QUE OS REGISTRADORES PRIORITÁRIOS AO FUNCIONAMENTO DA
;   MÁQUINA, E QUE PODEM SER ALTERADOS TANTO DENTRO QUANTO FORA DAS INTS
SEJAM
;  SALVOS EM REGISTRADORES TEMPORÁRIOS PARA PODEREM SER POSTERIORMENTE
;  RECUPERADOS.


SALVA_CONTEXTO
        MOVWF       W_TEMP          ; COPIA W PARA W_TEMP
        SWAPF       STATUS,W
        MOVWF       STATUS_TEMP     ; COPIA STATUS PARA STATUS_TEMP
        MOVF        PCLATH,W        ; MOVE PCLATH PARA O REGISTRADOR W
        MOVWF       PCLATH_TEMP     ; COPIA W PARA PCLATH_TEMP

        BTFSC       PIR1,RCIF       ; VERIFICA SE FOI INTERRUPÇÃO DE RECEPÇÃO
                                    ; PELA USART
        GOTO        CHEGOU_DADO     ; SIM - PULA PARA CHEGOU DADO
                                    ; NÃO

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *              FIM DA ROTINA DE INTERRUPÇÃO                *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  RESTAURAR OS VALORES DE "W" E "STATUS" ANTES DE RETORNAR.

SAI_INTERRUPCAO
```

```
        MOVF        PCLATH_TEMP,W
        MOVWF       PCLATH              ; COPIA PCLATH_TEMP PARA PCLATH
        SWAPF       STATUS_TEMP,W
        MOVWF       STATUS              ; COPIA STATUS_TEMP PARA STATUS
        SWAPF       W_TEMP,F
        SWAPF       W_TEMP,W            ; COPIA W_TEMP PARA W
        RETFIE                          ; RETORNA DA INTERRUPÇÃO

CHEGOU_DADO
        BANK0
        MOVF        RCREG,W             ; COLOCA EM REG W O DADO QUE CHEGOU
        MOVLW D'5'
        MOVWF PORTD

        BANK1
        BTFSS       TXSTA,TRMT          ; O BUFFER DE TX ESTÁ VAZIO ?
        GOTO        $-1                 ; NÃO - AGUARDA ESVAZIAR

        BANK0

        MOVWF        TXREG              ; COLOCA O DADO RECEBIDO NO REG DE
                                        ; TRANSMISSAO
        GOTO        SAI_INTERRUPCAO

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **********
; *         CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE      *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **********
;  NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
;  CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A
;  MÁQUINA E AGUARDA O ESTOURO DO WDT.

CONFIG
        CLRF        PORTD               ; LIMPA PORTA D
        CLRF        PORTC               ; LIMPA PORTA C
        BANK1                           ; SELECIONA BANCO 1 DA RAM

        MOVLW       B'01111111'
        MOVWF       TRISC               ; CONFIGURA O  I/O DO PORTC

        MOVLW       B'10000000'
        MOVWF       TRISD               ; CONFIGURA O  I/O DO PORTC

        MOVLW       B'00100100'
        MOVWF       TXSTA               ; CONFIGURA USART
                                        ; HABILITA TX
                                        ; MODO ASSINCRONO
                                        ; TRANSMISSÃO DE 8 BITS
                                        ; HIGH SPEED BAUD RATE

        MOVLW       D'23'
        MOVWF       SPBRG               ; ACERTA BAUD RATE -->9600bps

        MOVLW       B'11000000'
        MOVWF       INTCON              ; CONFIGURA INTERRUPÇÕES

        MOVLW       B'00100000'
```

```
        MOVWF       PIE1                ; NÃO HABILITA INTERRUPÇÃO DE
                                        ; TRANSMISSÃO DA USART
                                        ; HABILITA INTERRUPÇÃO DE RECEPÇÃO DA
                                        ; USART

        BANK0                           ; SELECIONA BANCO 0 DA RAM

        MOVLW       B'01111111'
        MOVWF       PORTD               ; COLOCA O VALOR  O1111111 NO PORTD

        MOVLW       B'10010000'
        MOVWF       RCSTA               ; CONFIGURA USART
                                        ; HABILITA RX
                                        ; RECEPÇÃO DE 8 BITS
                                        ; RECEPÇÃO CONTINUA
                                        ; DESABILITA ADDRESS DETECT
        BANK1

        MOVLW       B'11011011'
        MOVWF       OPTION_REG          ; CONFIGURA OPTIONS
                                        ; DESABILITA PULL-UPS
                                        ; INTERRUPÇÃO NA BORDA DE SUBIDA DO RB0
                                        ; O TMR0 SERÁ INCREMENTADO PELO CICLO DE
                                        ; MÁQUINA
                                        ; O PRESCALER SERÁ APLICADO AO WDT
                                        ; O WDT - SERÁ NA RAZÃO DE 1:8 E O TIMER 1:1

        BANK0

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *   AS INSTRUÇÕES A SEGUIR  FAZEM COM QUE O PROGRAMA TRAVE  QUANDO                    *
; * HOUVER UM RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT.               *
; * DESTA FORMA, SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA                  *
; * UM ESTOURO DE WDT E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO                      *
; *  START-UP DO PIC.                                                                  *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        BTFSC        STATUS, NOT_TO     ; RESET POR ESTOURO DE WDT ?
        GOTO        $                   ; NAO - CONTINUA NA MESMA LINHA
                                        ; (AGUARDA ESTOURO DO WDT);
                                        ; SIM

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                               INICIALIZAÇÃO DA RAM                                 *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0x20 A 0x70.
; EM SEGUIDA, AS VARIÁVEIS DE RAM DO PROGRAMA SÃO INICIALIZADAAS.

        MOVLW       0X20
        MOVWF       FSR                 ; APONTA O ENDEREÇAMENTO INDIRETO PARA
                                        ; A PRIMEIRA POSIÇÃO DA RAM

LIMPA_RAM
        CLRF        INDF                ; LIMPA A POSIÇÃO
        INCF        FSR,F               ; IMCREMENTA O PONTEIRO P/ A PRÓX. POS.
```

```
        INCF        FSR,W
        XORLW       0X80                ; COMPARA O PONTEIRO COM A ÚLT. POS. + 1
        BTFSS       STATUS,Z            ; JÁ LIMPOU TODAS AS POSIÇÕES ?
        GOTO        LIMPA_RAM           ; NÃO – LIMPA A PRÓXIMA POSIÇÃO
                                        ; SIM


; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                 LOOP PRINCIPAL                  *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

LOOP
        CLRWDT                          ; LIMPA WDT PARA NÃO ESTOURAR (NAO
                                        ; HAVER RESET)

        BTFSS       PORTD,7             ; PARTE DE TESTE DE PROGRAMA
        GOTO        SETA
        MOVLW       B'01111111'
        MOVWF       PORTD
        GOTO        LOOP
SETA
        MOVLW       B'00000000'
        MOVWF       PORTD
        GOTO        LOOP                ; FIM DA PARTE DE TESTE DE PROGRAMA


; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                 FIM DO PROGRAMA                 *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

END                                     ; FIM DO PROGRAMA
```

Apêndice B: Arquitetura da Rede INFINET

A configuração do SDCD que está instalado no parque industrial de Ubu é constituída por PCU's (unidade central de processamento), OIS's (estações de interface com operador), OIC's (consoles de interface com operador) e EWS's (estações de trabalho de engenharia), todos interligados por uma rede de comunicação chamada de INFINET.

As PCU's são instaladas próximas aos equipamentos que serão comandados, distribuindo assim o controle pela área. As PCU's recebem os sinais desses equipamentos da área (sinais de entrada), processam essas informações através de um programa (lógica) que é instalado nas mesmas, e mandam de volta aos equipamentos informações e comandos resultantes de seu processamento (sinais de saída), logo as PCU's podem ser consideradas, então, "computadores" responsáveis pela lógica de controle dos equipamentos do processo produtivo.

As OIS's são os computadores responsáveis pela interface do sistema de controle com os operadores da sala de controle - interface homem-máquina. Através das OIC's, que são ligadas as OIS's, os operadores recebem informações do processo e atuam nos equipamentos controlando-os.

A rede de comunicação (proprietária) é a INFINET. Está rede é responsável pela troca de informações entre todos os seus nós, que são as PCU's, OIS's e EWS's, permitindo, assim, a integração de todo o sistema.

Outra característica marcante do SDCD é a redundância do hardware. Os equipamentos principais, como cartões de comunicação, cartões de processamento da lógica, cabos de comunicação da rede INFINET e estações de operação possuem redundância, ou seja, os equipamentos são duplicados, permitindo uma maior confiabilidade ao sistema.

Na figura 28 é apresentada a arquitetura da rede INFINET atualmente instalada em Ponta Ubu, a qual é constituída de três LOOP's (anéis) interligados entre si, onde estão os nós de comunicação com as PCU's, OIS's e EWS's. Tem-se também uma rede local *Ethernet*, onde interligamos as Consoles de Operação (OIC's) com as Estações de Operação (OIS's), responsáveis pela interface Homem-Máquina nas salas de controle. A arquitetura interna de uma PCU está representada na figura 29 do relatório.

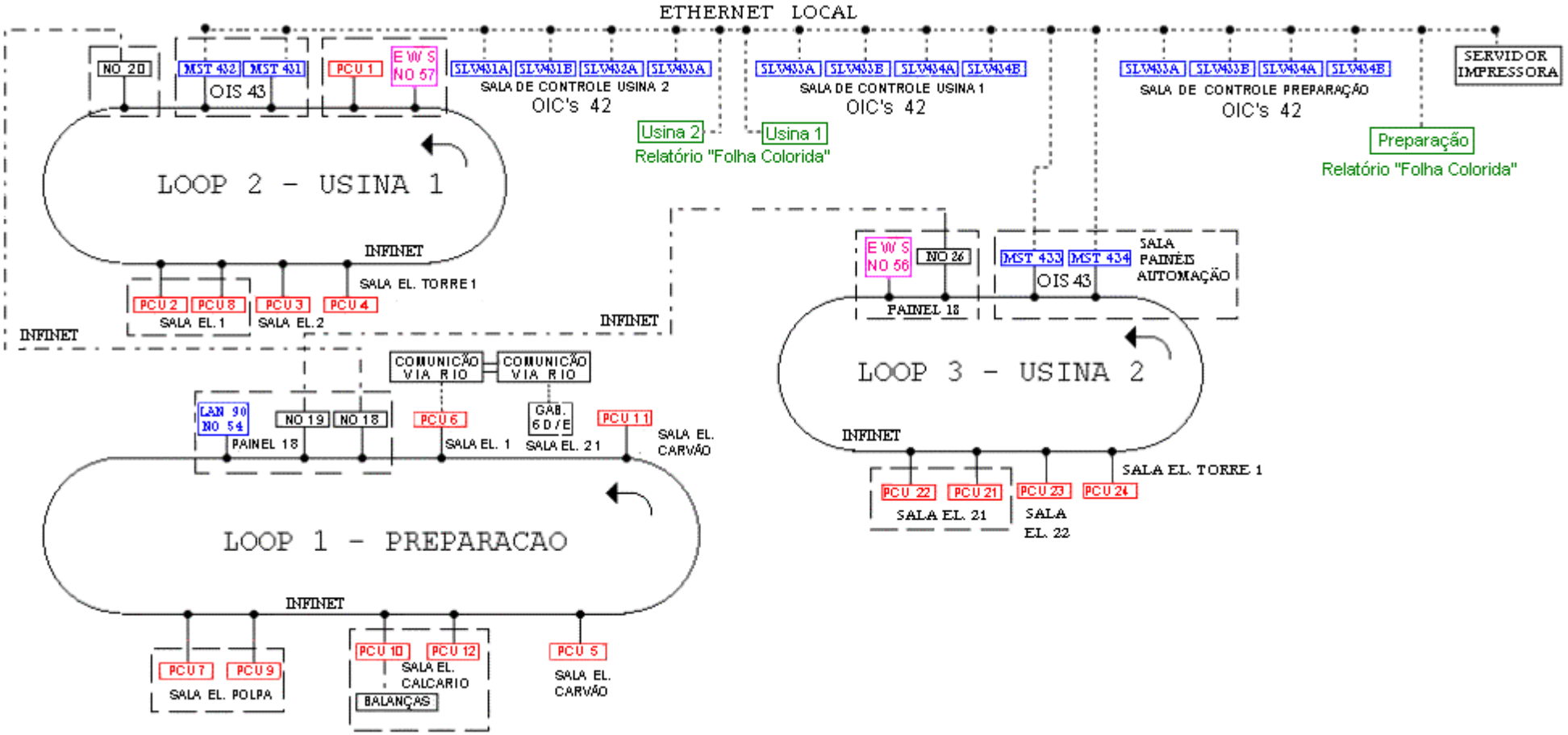

Figura 28: Arquitetura da Rede Infinet

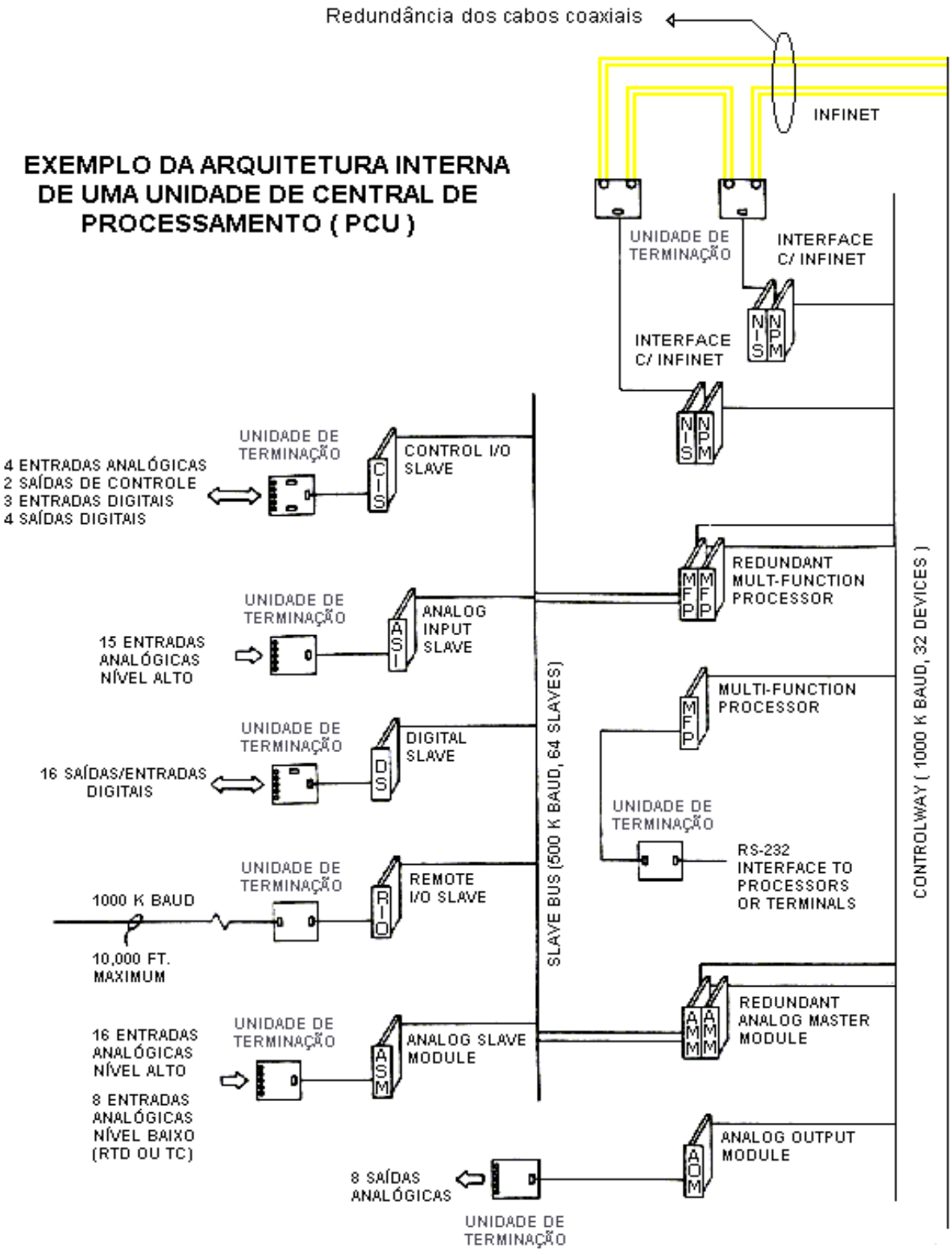

Figura 29: Arquitetura Interna de uma PCU.

Apêndice C: Sistema de Controle Otimizante

O OCS é um sistema de controle especialista avançado utilizado para controlar e supervisionar a planta. Os objetivos previstos para o OCS são:

- Gerenciamento de condições anormais de operação. Isto inclui detecção e ações de correções para alta resistência (baixa permeabilidade) da camada de pelotas e para a temperatura excessiva da pelota no fim da zona de resfriamento;
- Otimização do processo quando de condições normais de operação, objetivando um menor consumo de energia levando-se em conta a permanência da boa qualidade das pelotas (acima do mínimo requerido) e outras restrições de produção que devam ser consideradas para melhor atendimento dos clientes da empresa.

Além destes objetivos, o OCS possibilita a diminuição de consumo do óleo combustível, o aumento da produção, uma melhor administração do fluxo de gases e a diminuição da energia específica. Estes benefícios são alcançados, pois o OCS gera uma recuperação mais rápida do processo em caso de situações anormais e será operado de forma otimizada em situações normais.

O que se percebe é que através destes benefícios ocorrerá uma maior disponibilização de novas informações para os engenheiros de processo e operadores, incluindo valores de processo inferidos pelo modelo e conclusões de regras avançadas de diagnósticos.

A estratégia de controle implementada pelo OCS continuamente adapta os *set points* do processo para atingir os objetivos anteriormente definidos e envia mensagens aos operadores com sugestões de ações que não são implementadas automaticamente.

Junto ao sistema de controle otimizante está incluído um conjunto de regras para verificar o *status* da instrumentação e decidir como lidar com várias situações quando diagnosticado que um ou mais sensores não estiverem funcionando adequadamente. Este conjunto de regras nada mais é que um sistema de controle *Fuzzy* especialista desenvolvido pelos engenheiros e operadores da empresa.

Apêndice D: Fluxogramas do módulo Protocolo e explicações detalhadas

A primeira parte do módulo do Protocolo se resume às ligações feitas entre os blocos do fluxograma da figura abaixo.

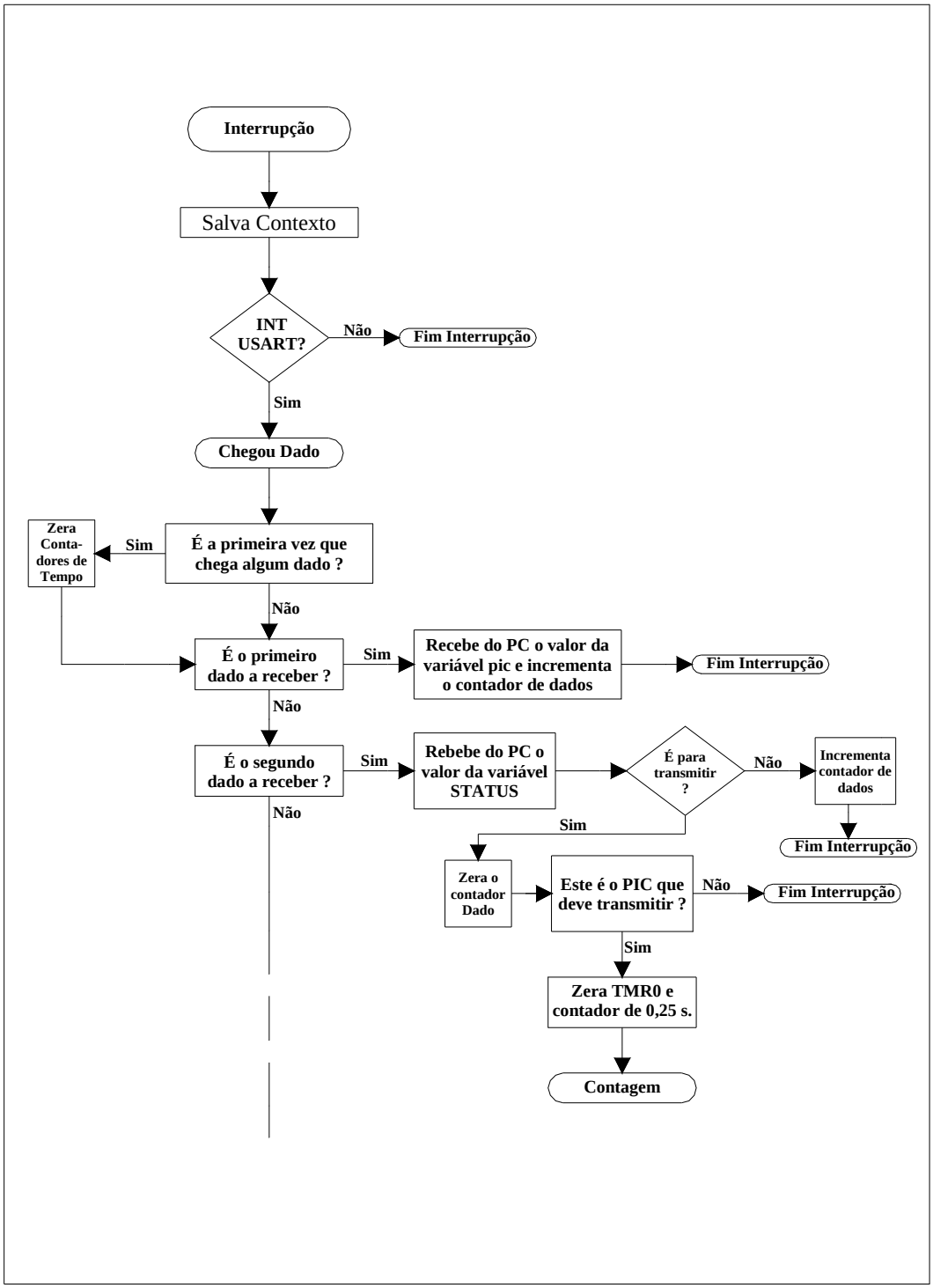

Figura 30: Fluxograma da parte inicial do módulo 'Protocolo'

Do fluxograma acima, vê-se que esta rotina só ocorre quando há comunicação (envio e recebimento de dados). Em seqüência são tratadas as variáveis ‘pic’ e ‘STATUS’ solicitando a transmissão ou recepção de dados pelo microcontrolador. A parte descontínua na figura 30 indica a continuidade do módulo Protocolo. A continuidade é representada na figura abaixo.

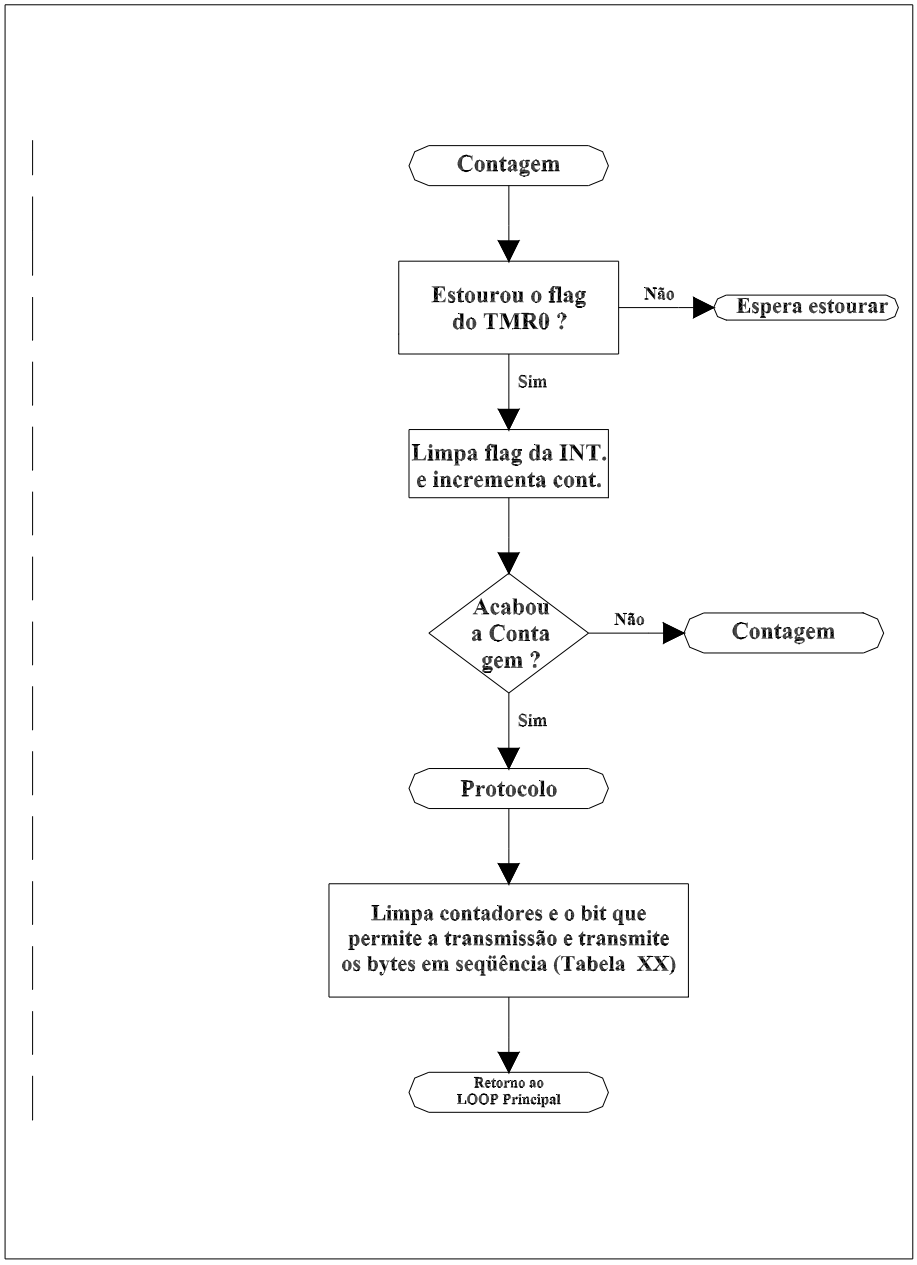

Figura 31: Fluxograma da segunda parte do módulo ‘Protocolo’

A segunda parte do protocolo mostra o tratamento dado pelo módulo quando é solicitada a transmissão de dados do PIC para o computador.

O computador permanece constantemente transmitindo informações do sistema de controle de temperatura para os PIC’s de forma alternada, ou seja, primeiro solicita o PIC 1 e

depois solicita os demais PICs do sistema (nunca ao mesmo tempo – evitar erros de transmissão) a fazerem suas transmissões e operações de controle. O que determina se o PIC deve transmitir ou não ao computador é o valor da variável Status (memória RAM) que após setado possibilita tal transmissão. Para evitar que os microcontroladores fiquem enviando inúmeros dados repetidos e desnecessários, criou-se uma sub-rotina utilizando o timer 0 que realiza a contagem de 0,25s e depois desloca-se para a rotina Protocolo onde ocorre a transmissão dos dados. Mesmo porque o *Ploter* do supervisório só apresenta diferença de um dado para outro com aproximadamente 0,5s (projeto no *software* desenvolvido em Java).

Dentre as variáveis enviadas do PIC ao PC há uma certa peculiaridade relacionada às variáveis TEMP_SALA1, TEMP_SALA2, TEMP_SALA3 e TEMP_SALA4 enviadas dos PICs ao computador. O microcontrolador 1 que possui a variável NUMPIC constante e igual a 1 envia na verdade TEMP_SALA1_LO (variável da sala em que têm um dos atuadores (secadores) → necessidade de bytes LO e HI para tratamento da variável no controle PID), TEMP_SALA2, o valor zero e TEMP_EXT_1, respectivamente. Enquanto que o microcontrolador 2 que possui a variável NUMPIC constante e igual a 2 envia ao PC TEMP_SALA3, TEMP_SALA4, TEMP_SALA5_LO (variável da sala em que têm um dos atuadores → necessidade de bytes LO e HI para tratamento da variável na rotina de controle PID) e TEMP_EXT_2. Estas variáveis de temperatura das salas representam exatamente o número em que elas foram mostradas na figura 06 da maquete.

Da linha descontínua da segunda parte do protocolo segue a recepção dos valores enviados do computador aos microcontroladores. Antes de receber qualquer valor, os PIC's verificam se o valor enviado é para eles mesmo, através da conferência do valor da variável 'pic', evitando assim que valores de variáveis de outros PICs sejam atribuídos de forma equivocada a eles, e incrementam o contador para recebimento das próximas variáveis enviadas pelo computador. Esta terceira parte do módulo do protocolo pode ser vista na figura 32.

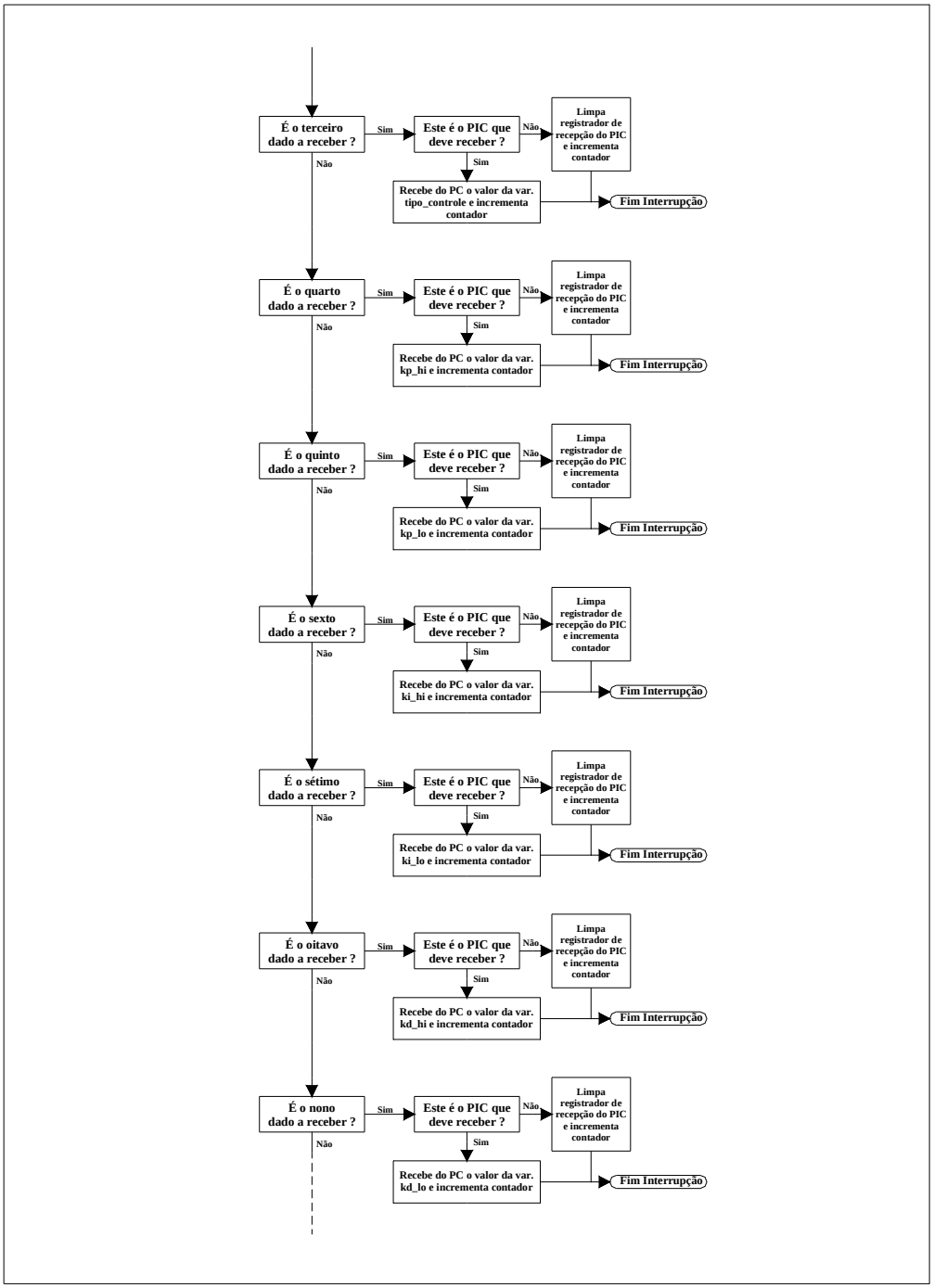

Figura 32: Fluxograma da terceira parte do módulo 'Protocolo'

Para finalizar o módulo Protocolo os microcontroladores continuam tratando os valores enviados pelo computador como na terceira parte do módulo. Apenas no último dado a ser recebido por um dos PIC's é que o sistema pergunta se é para sair do programa ou continuar. Em caso de um pedido de saída feito pelo usuário, o programa desloca para uma rotina na qual se espera o pedido de reinicialização do programa e zera os contadores de tempo. Em caso de não haver um pedido de saída do programa é atribuído zero ao contador de

dados recebidos pelo PIC para que possa se reiniciar o processo novamente. Estas informações podem ser verificadas na figura 33 abaixo.

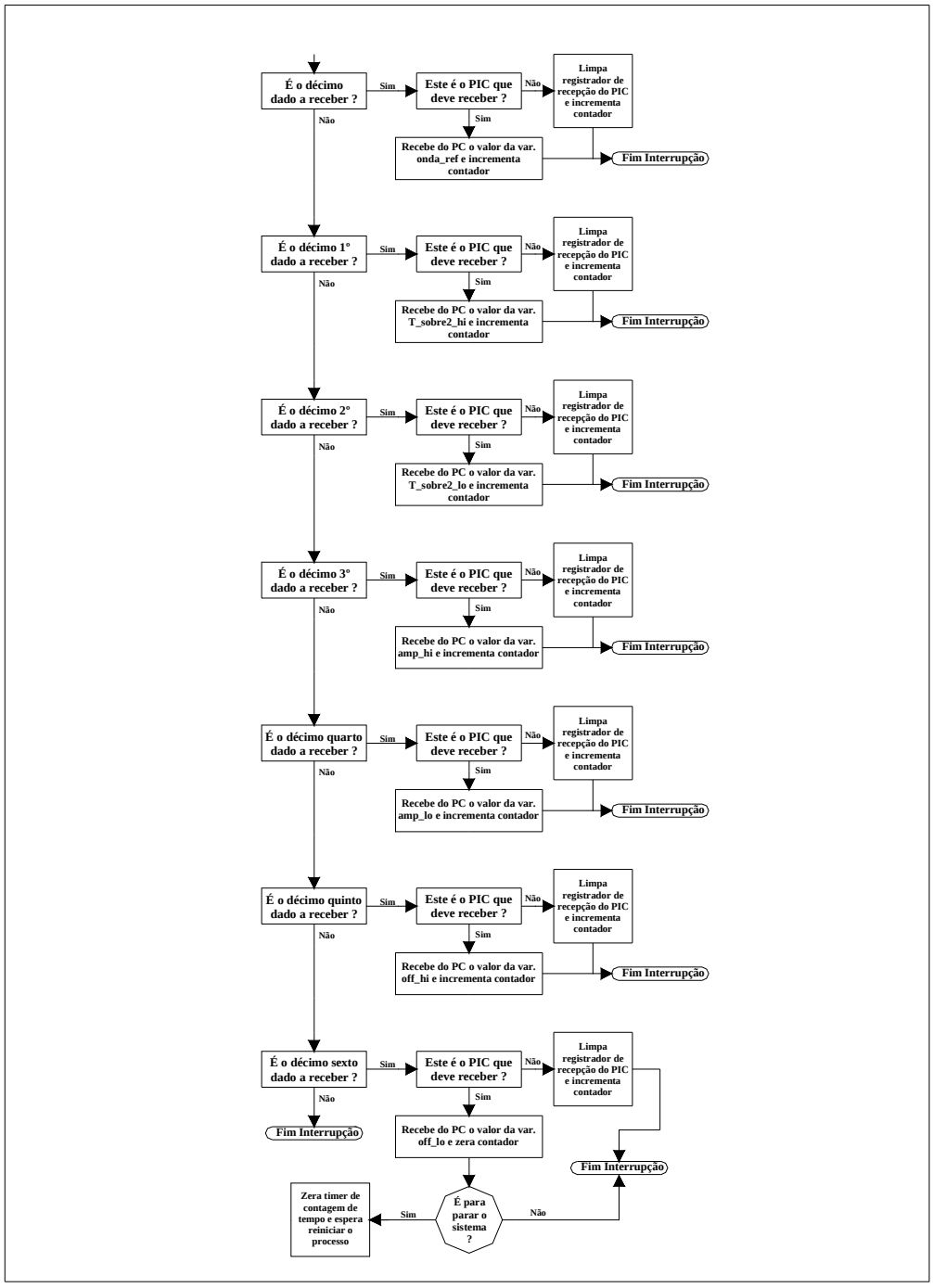

Figura 33: Fluxograma da parte final do módulo 'Protocolo'

Apêndice E: Fluxograma do módulo "Onda Quadrada" e explicações detalhadas

Nesta subrotina do programa principal, através dos dados enviados pelo usuário (amplitude, freqüência e *offset*), criou-se a onda quadrada que possui os eixos Tempo X Amplitude, da seguinte forma:

A cada meio período (T_sobre2) o valor da onda de referência é alterado na subrotina 'MUDA_REF'. Nesta subrotina ocorre a comparação entre o valor da referência e os níveis alto e baixo da onda. Caso o valor de referência seja igual ao nível alto, por exemplo, a subrotina chama outra subrotina (neste caso REF_BAIXO), onde é a atribuído o valor de referência → Referência = Offset - Amplitude/2. Se por outro lado o valor de referência seja igual ao nível baixo, subrotina chama outra subrotina (neste caso REF_ALTO), onde é a atribuído o valor de referência → Referência = Offset + Amplitude/2. O fluxograma do módulo Onda Quadrada é apresentado na figura 34 abaixo

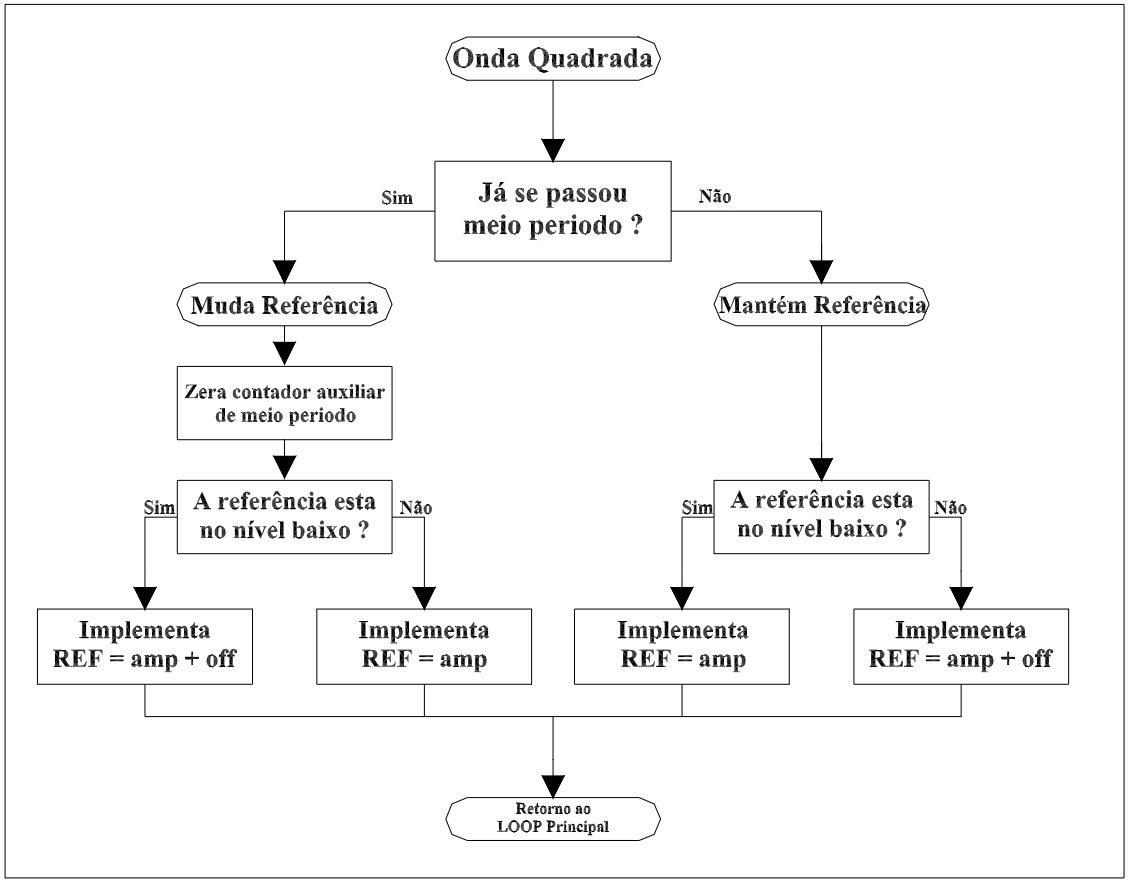

Figura 34: Fluxograma do módulo "Onda Quadrada"

Nos intervalos entre os meio períodos a rotina deslocasse para a subrotina ‘MANTEM_REF’, onde simplesmente compara os bytes de referência com os níveis alto e baixo e atribui valores iguais aos níveis encontrados na verificação.

Apêndice F: Fluxograma do módulo "Onda Triangular" e explicações detalhadas

Da mesma maneira que a rotina da onda quadrada, no módulo da onda triangular recebe-se os valores desejados pelo usuário de amplitude, freqüência e *offset* e gera-se a onda triangular.

Esta onda é gerada da forma apresentada na figura 35 abaixo.

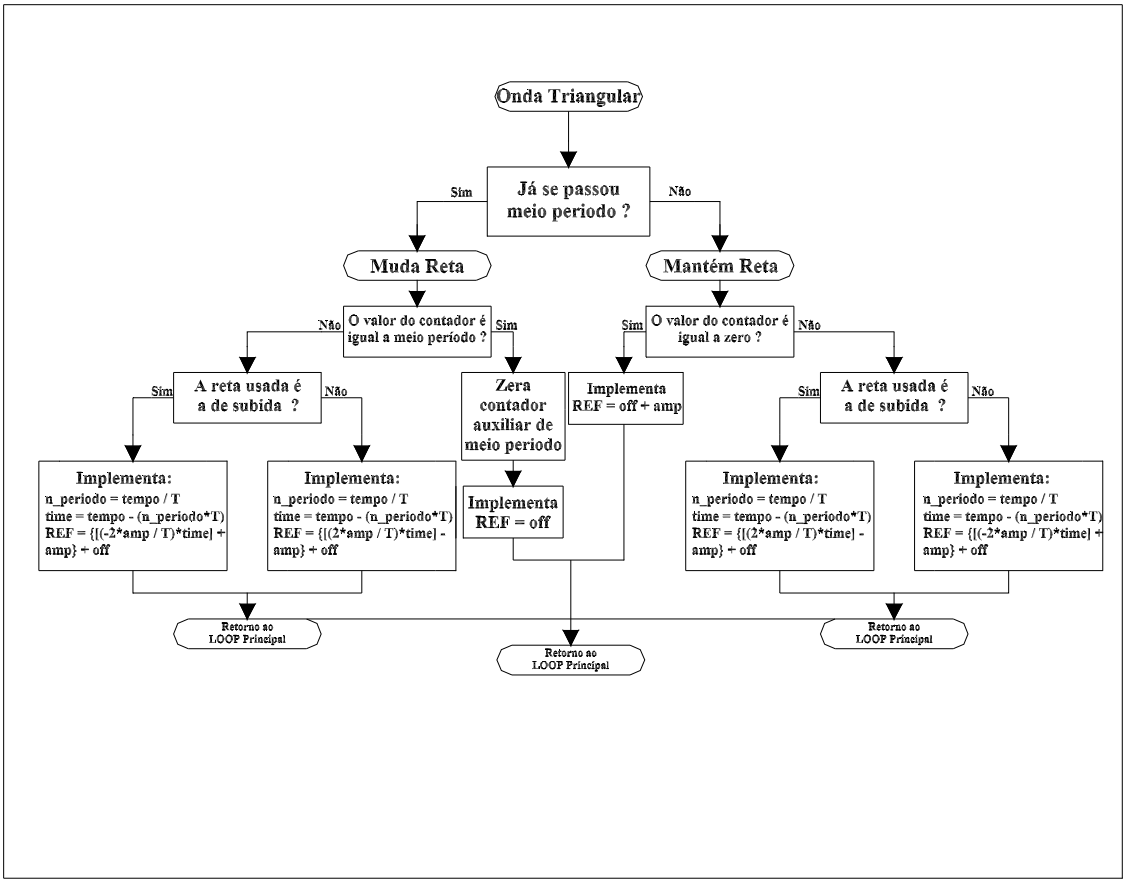

Figura 35: Fluxograma do módulo "Onda Triangular"

Algo de novo nesta onda é a contagem de um tempo auxiliar para facilitar a montagem da onda triangular. As equações de formação da onda triangular são as seguintes:

- REF = (-2*amp*freq*time + amp) + offset → Reta de subida
- REF = (2*amp*freq*time + amp) – offset → Reta de descida

Obs: time ≠ TEMPO.

A cada meio período (T_sobre2) ocorre à mudança de equação e nos intervalos entre um período e outro se mantém as equações em uso. Para que o tempo (time) não cresça

indefinidamente como o contador de tempo do experimento (TEMPO) e provoque erros na formação da onda triangular, o tempo auxiliar informado anteriormente é utilizado.

Da expressão:

- N_periodo = TEMPO / T
- time = TEMPO – (N_periodo*T)

Faz-se a atualização do temporizador (time = tempo auxiliar), não interfere na contagem de tempo do experimento e gera-se corretamente a onda.

Apêndice G: Fluxograma do módulo “Onda Degrau” e explicações detalhadas

Neste tipo de onda o usuário só informa o valor de amplitude que deseja e a rotina quando chamada atribui o valor enviado pelo usuário a referência provocando um degrau de amplitude como se vê na figura abaixo.

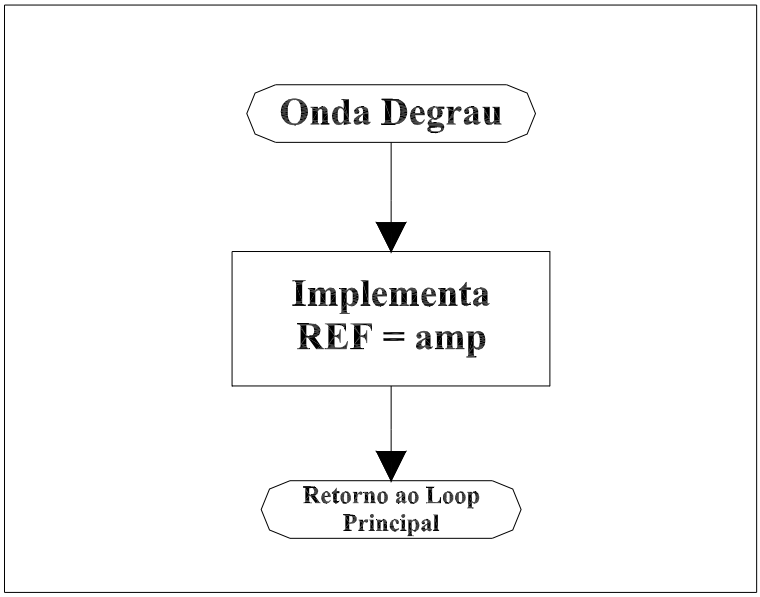

Figura 36: Fluxograma do módulo “Onda Degrau”

Apêndice H: Principais Características do PIC16F877A

A maior parte deste Apêndice foi retirada de [14] e de [15].

Principais características

- CPU RISC de alto desempenho;
- 35 instruções de 8 bits, que usam apenas um ciclo para execução (menos as que executam desvios);
- Velocidade de operação de até 20MHz;
- Memória dividida em 3 partes:
  - Memória de programa FLASH: 8 K de 14 bits (instrução);
  - Memória de dados RAM volátil: 368 bytes;
  - Memória de dados não-volátil (EEPROM): 256 bytes.
- Possibilidade de uso de até 14 fontes de interrupção:
  - Interrupção de Timer 0, Timer 1 e Timer 2 ;
  - Interrupção externa (sinal externo ligado ao pino RB0);
  - Interrupção por mudança de estado (sensível a diferença de nível existente entre o pino e o *latch* interno – portas RB4, RB5, RB6 e RB7);
  - Interrupção da porta paralela (PSP);
  - Interrupção dos conversores A/D;
  - Interrupção de recepção e transmissão da USART (*Universal Synchronous Asynchronous Receiver Tansmitter*);
  - Interrupção de comunicação serial (SPI e I^2C);
  - Interrupção do CCP1 e CCP2 (*Capture/Compare/PWM*);
  - Interrupção de fim de escrita na EEPROM/FLASH;
  - Interrupção de colisão de dados (*Bus Collision*).
- Oscilador selecionado dentre 4 opções: ressoador, cristal, RC ou circuitos de oscilação (híbridos);
- Proteção de código programável.

Características periféricas básicas

- 3 contadores (*timers*) com prescaler, um de 16 bits (TMR1) e dois de 8 bits (TMR0 e TMR2);
- 2 módulos que servem para captura, comparação e geração de sinal PWM (resolução de 10 bits);
- Conversor analógico-digital embutido, possibilitando o uso de até 8 canais de entrada com resolução de 10 bits;
- Uso de até 33 pinos para operações de entrada e saída.

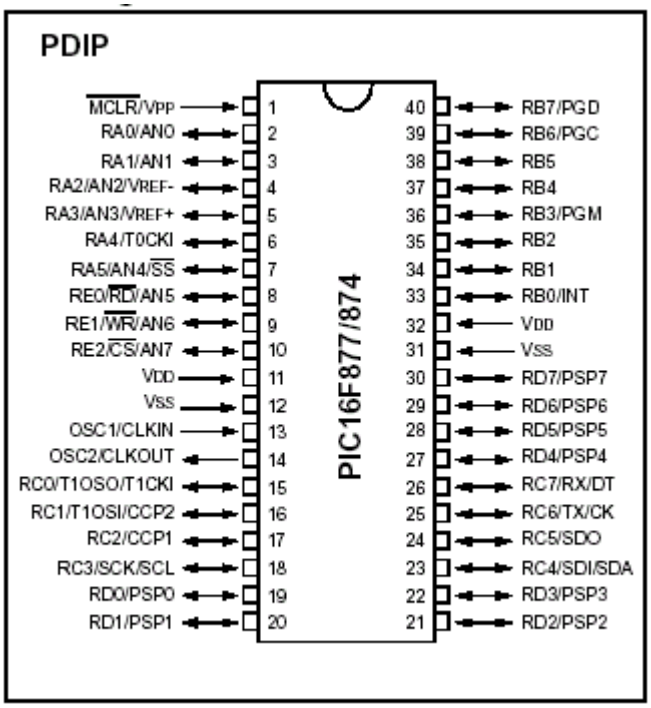

Figura 37: Relação de pinos do PIC16F877 (mesma relação para o PIC16F877A) , encapsulamento PDIP

Organização da memória

Como foi dito anteriormente, existem três blocos de memória nos PIC´s: programa, dados e EEPROM. Os dispositivos PIC16F87XA possuem um contador de programa de 13 bits capaz de endereçar um espaço de memória de 8Kx14 bits. Dois endereços desta memória de programa são reservados para duas tarefas importantes: o endereço “0x0000”, para “reset” e o endereço “0x0004” para interrupções.

A memória de dados é particionada em 4 “bancos” de 128 bytes que contém dois tipos de registradores classificados de acordo com sua utilização: registradores de uso geral e

registradores para funções especiais. O acesso a estes "bancos" é feita através da escrita nos bits 5 e 6 do registrador "STATUS".

A memória de dados EEPROM pode ser lida e escrita durante a operação do PIC. Está memória pode ser apagada eletricamente e regravada (gravada com gravadores específicos ou pelo sistema).

Estas operações são feitas usando apenas um byte desta memória. Uma escrita ocasiona uma operação para apagar e então escrever o novo valor, não causando nenhum impacto sobre o dispositivo, diferentemente de uma escrita na memória de programa. A memória de dados EEPROM mostra-se bastante útil, portanto, para armazenar resultados mesmo quando o dispositivo está sem alimentação (desligado).

Principais Registradores de Funções Especiais (SFR)

Estes registradores são usados pela CPU e pelos módulos periféricos para controlar o dispositivo para que o mesmo opere da forma desejada. Implementados como memória RAM estática, podem ser classificados em dois tipos: "core" (CPU) e periféricos. A seguir são citados os principais registradores SFR:

- STATUS: este SFR contém o estado aritmético da ULA (unidade de lógica e aritmética), se o dispositivo foi "resetado" e os bits para seleção dos bancos da memória de dados.
- OPTION: serve para configuração de várias opções para operação do microcontrolador. Contém bits para controle do "*prescaler*" do contador TMR0, controle do postscaler do WDT (w*atch dog timer*).
- INTCON: usado para identificação e configuração de diversas interrupções.
- PIE1, PIE2, PIR1, PIR2: contêm bits individuais para habilitação de interrupções periféricas (eventos externos).
- PCON: registrador para possibilitar diferenciação entre diversos tipos de "RESET" que podem ser acionados.
- PCL e PCLATH: o PCL armazena os 8 bits menos significativos do contador de programa (PC). O PCLATH armazena os 5 bits mais significativos do contador de programa.
- ADCON0 e ADCON1: registradores para configuração do módulo AD, além da seleção de 1 entre de 8 canais a ser recebido. O resultado da conversão de 10 bits é

armazenado no par de registradores ADRESH (AD result high) e ADRESL (AD result low).

- CCP1CON e CCP2CON: usados para configurar os módulos PWM. Em conjunto com o registrador T2CON, definem o período do sinal PWM e o período do ciclo de trabalho, que deve ser armazenado no registrador CCPR1L (8 bits mais significativos) e nos bits 4 e 5 do registrador CCP1CON (2 bits menos significativos), totalizando os 10 bits de resolução do módulo PWM.

Os registradores a seguir também são SFR, mas são utilizados para uso das portas de E/S do PIC:

- TRISA, TRISB, TRISC, TRISD e TRISE: usados para configurar os pinos das portas: PORTA, PORTB, PORTC, PORTD e PORTE; respectivamente como entrada (1) ou saída (0).
- Os registradores PORT, por sua vez, contêm os dados as serem enviados pelo PIC quando configurados como saída ou os dados a serem recebidos quando configurados como entrada.

Especificações a respeito de detalhes sobre operação dos módulos internos do PIC16F877A e demais registradores SFR encontram-se no *data-sheet* do dispositivo.

Apêndice I: Código Java do Programa "Servidor"

```
/*****************************************************************************************
Nome do programa: Servidor
Autor: Alexandre Silva Souza
Data de início: 25/03/03
Ultima modificação:07/07/03
Finalidade: Fazer comunicação com o microcontrolador PIC16F877A
        através da porta serial (Conector DB9) do PC.
*****************************************************************************************/

import java.util.*;
import javax.comm.*;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.text.DecimalFormat;

public class sc extends JFrame
{
   byte dado, ref1_B,numPic_B, numCont_B,tipoCont_B,
        Sala1Pic_B,Sala2Pic_B,Sala3Pic_B,TambPic_B;
   Byte dadoByte;
   int Istatus;
   boolean Bprotocol;

   private short TempoPic_S,RefPic_S,AtuPic_S,
        amp1_S,fre1_S,ofs1_S,p1_S,i1_S,d1_S;

   private JTextField enter;
   private JTextArea display;
   private Conexao conexao;
   private rodaServidor rodaservidor;
   private InputStream inputStream;
   private OutputStream outputStream;
   private DataOutputStream dataOutputStream_Pic,dataOutputStream_App;
   private DataInputStream dataInputStream_Pic,dataInputStream_App;
   private DecimalFormat formata;
   public final byte Pic_1 =1;
```

```
   public final byte Pic_2 =2;
   public final byte PICTRA = 1;
   public final byte PICREC = 2;
   public final byte PARAR = 4;
   public final byte ZERO = 0;

   String Pacote = "";

public sc()
{
   super( "Servidor Térmico" );

   Container c = getContentPane();

   formata = new DecimalFormat("000.00");

   enter = new JTextField();
   enter.setEnabled( true );
   enter.addActionListener(
      new ActionListener() {
         public void actionPerformed( ActionEvent e )
         {
            Byte b = new Byte(e.getActionCommand());
            byte x = b.byteValue();
            sendPic(x);  //Envia string pela porta serial

         }
      }
   );
   c.add( enter, BorderLayout.NORTH );
   display = new JTextArea();
   c.add( new JScrollPane( display ),
      BorderLayout.CENTER );

   setSize(300, 150 );
   show();
}
public synchronized void sendProtocol(byte Pic,byte funcao)
{
   try
```

```
   {
      dataOutputStream_Pic.writeByte(Pic);
      display.append("Destino PIC: " + String.valueOf(Pic)+ "\n");

      dataOutputStream_Pic.writeByte(funcao);
      display.append("Status: " + String.valueOf(funcao)+ "\n");

      //Tipo de controle
      dataOutputStream_Pic.writeByte(tipoCont_B);
      display.append("Tipo Controle: " + String.valueOf(tipoCont_B)+ "\n");

      //Enviando para o Pic dados referêntes ao controlador
      dataOutputStream_Pic.writeShort(p1_S);  //Send the high byte fist
      display.append("Kp: " + String.valueOf(p1_S)+ "\n");
      dataOutputStream_Pic.writeShort(i1_S);
      display.append("Ki: " + String.valueOf(i1_S)+ "\n");
      dataOutputStream_Pic.writeShort(d1_S);
      display.append("Kd: " + String.valueOf(d1_S)+ "\n");

      //Tipo de referência (1 senoide) (2 quadrada) (4 triangular) (8 degrau)
      dataOutputStream_Pic.writeByte(ref1_B);
      display.append("TipoReferência: " + String.valueOf(ref1_B)+ "\n");

      //Propriedades da referência
      dataOutputStream_Pic.writeShort(fre1_S);
      display.append("Frequência: " + String.valueOf(fre1_S)+ "\n");
      dataOutputStream_Pic.writeShort(amp1_S);
      display.append("Amplitude: " + String.valueOf(amp1_S)+ "\n");
      dataOutputStream_Pic.writeShort(ofs1_S);
      display.append("Offset: " + String.valueOf(ofs1_S)+ "\n");
   }
   catch (IOException e)
   {
      display.append("Erro ao enviar dados para o PIC");
   }
}
private void sendApp(String s)
{
   try
   {
```

```
            dataOutputStream_App.writeUTF( s );
            dataOutputStream_App.flush();
            display.append("Dados enviado para o Applet \n \n" + s + "\n");
        }
        catch (IOException cnfex)
        {
            display.append("\n Erro ao enviar dados para o Applet");
        }
    }
    private synchronized void sendPic(byte pic)
    {
        try
        {
            dataOutputStream_Pic.writeByte(pic);
            dataOutputStream_Pic.writeByte(PICTRA);
            display.append("Pedido de Transmissão ao Pic: " +String.valueOf(pic)+ "\n");
            display.append("Status Pic: " +String.valueOf(PICTRA)+ "\n");
            //display.append("Dado enviado: " +getBits(dado)+ "\n");
        }
        //catch (NumberFormatException x)
        //{}
        catch (IOException e)
        {
            display.append("Erro ao fazer pedido de transmissão ao Pic: " + String.valueOf(pic) + "\n");
        }
    }
    private void rodaconexao()
    {
        conexao = new Conexao();
        rodaservidor = new rodaServidor();
    }
    public static void main( String args[] )
    {
        sc app = new sc();

        app.addWindowListener(
            new WindowAdapter() {
                public void windowClosing( WindowEvent e )
                {
                    System.exit( 0 );
```

```
        }
      }
   );
   app.rodaconexao();
}

/************************************************************************
Descrição: "Conexão" classe interna que faz toda a programação necessária
para obter, abrir, enviar e receber dados pela porta serial.
************************************************************************/

private class Conexao implements Runnable, SerialPortEventListener
{

CommPortIdentifier portId;
Enumeration portList;
SerialPort serialPort;
Thread readThread;
int numBytes;

public Conexao()
{
   portList = CommPortIdentifier.getPortIdentifiers();  //Retorna todas as portas conhecidas pelo sistema
   while (portList.hasMoreElements())                   //em uma vetor (Enumeration) de CommPortIdentifiers
   {
      portId = (CommPortIdentifier) portList.nextElement();
      if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) // Verifica se a porta é do tipo serial
      {
         if (portId.getName().equals("COM2"))  //Verifica se a porta serial é a COM2
         {
            abrePorta();
         }
      }
   }
}
private void abrePorta()  //Abre a porta serial
{
   try
   {
      serialPort = (SerialPort) portId.open("Comunicação com PIC", 2000);
```

```
        display.append("A porta COM2 foi aberta com sucesso!\n");
        configuraParametros();
    }
    catch (PortInUseException e)
    {
        display.append("Erro ao tentar abrir a porta COM2!");
    }
}
private void configuraParametros() // Configura os parametros de transmissão
{
    try
    {
        serialPort.setSerialPortParams(9600,
        SerialPort.DATABITS_8,
        SerialPort.STOPBITS_1,
        SerialPort.PARITY_NONE);
        obtemStreams();
    }
    catch (UnsupportedCommOperationException e)
    {
        display.append("Erro ao tentar configurar parâmetros da porta COM2!");
    }
}

private void obtemStreams() //Obtem os stream da transmissão
{
    try
    {
        inputStream = serialPort.getInputStream();
        dataInputStream_Pic = new DataInputStream(inputStream);
        outputStream = serialPort.getOutputStream();
        dataOutputStream_Pic = new DataOutputStream(outputStream);
        try
        {
            serialPort.addEventListener(this);
            serialPort.notifyOnDataAvailable(true);  //Expresses interest in receiving notification when
                                        //input data is available
            readThread = new Thread(this);
            readThread.start();
        }
```

```
        catch (TooManyListenersException e)
        {
           display.append("Erro ao tentar definir ouvinte de eventos da serial!");
        }
    }
    catch (IOException e)
    {
        display.append("Erro na obtenção dos Streams!");
    }
}
public void run()
{
    try
    {
        Thread.sleep(20000);  //Coloca o programa (Thread) para dormir
                        //Fica aguardando enventos, na porta serial ou na Interface com usuário
    }
    catch (InterruptedException e)
    {}
}
public void empacota()
{
    double TempoPic_D = TempoPic_S*35*0.001; // Corrige o tempo para segundos
    float Sala1Pic_I,Sala2Pic_I,Sala3Pic_I,TambPic_I;
    if (Sala1Pic_B < ZERO)
    {
        Sala1Pic_I = (127 -Sala1Pic_B)/2;
    }
    else
    {
        Sala1Pic_I = (Sala1Pic_B)/2;
    }
    if (Sala2Pic_B < ZERO)
    {
        Sala2Pic_I = (127 -Sala2Pic_B)/2;
    }
    else
    {
        Sala2Pic_I = (Sala2Pic_B)/2;
    }
```

```
   if (Sala3Pic_B < ZERO)
   {
      Sala3Pic_I = (127 -Sala3Pic_B)/2;
   }
   else
   {
      Sala3Pic_I =(Sala3Pic_B)/2;
   }
   if (TambPic_B < ZERO)
   {
      TambPic_I = (127 -TambPic_B)/2;
   }
   else
   {
      TambPic_I = (TambPic_B)/2;
   }

   Pacote = String.valueOf(numPic_B) + " " + formata.format(TempoPic_D).replace(',','.') + " " +
      String.valueOf(RefPic_S/2) + " " + String.valueOf(Sala1Pic_I) + " " +
      String.valueOf(Sala2Pic_I) + " " + String.valueOf(Sala3Pic_I) + " " +
      String.valueOf(TambPic_I) + " " + String.valueOf(AtuPic_S);
}
public void gravar_arquivo(String s)
{
   //File file = new File("y:\\","dadosPic.txt");
   try
   {
      FileWriter gravar_arq = new FileWriter("y:\\dados\\dadosPic.txt",true);
      gravar_arq.write(s + "\n");
      gravar_arq.close();
   }
   catch(IOException io)
   {
      display.append("\nErro ao gravar o arquivo");
   }
}

public void serialEvent(SerialPortEvent event)  //Método chamado quando ocorre eventos na porta serial
{
   switch(event.getEventType())
```

```
  {
     case SerialPortEvent.BI:  //Break interrupt
     case SerialPortEvent.OE:  //Overrun error
     case SerialPortEvent.FE:  //Framing error
     case SerialPortEvent.PE:  //Parity error
     case SerialPortEvent.CD:  //Carrier detect
     case SerialPortEvent.CTS:  //Clear to send
     case SerialPortEvent.DSR:  //Data set ready
     case SerialPortEvent.RI:  //Ring indicator
     case SerialPortEvent.OUTPUT_BUFFER_EMPTY:  //Output buffer is empty
        break;
     case SerialPortEvent.DATA_AVAILABLE:  //Data available at the serial port
        try
        {
           numPic_B = dataInputStream_Pic.readByte();
           display.append("1-Numero do Pic: "+String.valueOf(numPic_B)+"\n");
           if (numPic_B == Pic_1 || numPic_B == Pic_2)// Este parametro está sendo colocado devido a um ruido
que aparece quando
                       //liga o pic com o software executando. Neste caso aparece um dado com valor
                       //zero na porta!
           {
              TempoPic_S = dataInputStream_Pic.readShort();
              display.append("2e3-Tempo: "+String.valueOf(TempoPic_S)+"\n");
              RefPic_S = dataInputStream_Pic.readShort();
              display.append("4e5-Referência: "+String.valueOf(RefPic_S)+"\n");
              Sala1Pic_B = dataInputStream_Pic.readByte();
              display.append("6-Sala1: "+String.valueOf(Sala1Pic_B)+"\n");
              Sala2Pic_B = dataInputStream_Pic.readByte();
              display.append("7-Sala2: "+String.valueOf(Sala2Pic_B)+"\n");
              Sala3Pic_B = dataInputStream_Pic.readByte();
              display.append("8-Sala3: "+String.valueOf(Sala3Pic_B)+"\n");
              TambPic_B = dataInputStream_Pic.readByte();
              display.append("9-Temp. Amb: "+String.valueOf(TambPic_B)+"\n");
              AtuPic_S = dataInputStream_Pic.readShort();
              display.append("10e11-Atuador: "+String.valueOf(AtuPic_S)+"\n");

              if (Bprotocol) //É pra enviar o protocolo?
              {
                 if (Istatus == 0)
                 {
```

```
                    sendProtocol(Pic_1,PARAR);
                    sendProtocol(Pic_2,PARAR);
                }
                else
                    sendProtocol(numCont_B,PICREC);
                Bprotocol = false;
            }
            if (Istatus == 1)
            {
                empacota();
                sendApp(Pacote);
                gravar_arquivo(Pacote);

                if (numPic_B == Pic_1)
                {
                    sendPic(Pic_2);
                }
                else
                {
                    sendPic(Pic_1);
                }
            }
        }
    }
    catch (EOFException eof)
    {
        display.append("Erro ao receber bytes \"Short\" do PIC-Fim de Fluxo!");
    }
    catch (IOException e)
    {}
    break;
  }
}
} //Fim da classe interna "Conexão".

/*****************************************************************************
Descrição: "rodaServidor" classe interna que faz toda a programação necessária
para obter a conexao e enviar e receber dados para o usuário remoto (applet).
*****************************************************************************/
```

```
private class  rodaServidor implements Runnable
{
private ServerSocket server;
private Socket connection;
private int counter = 1;
private Thread esperaConexao;
private boolean PrimeiroPedido;

public rodaServidor()
{
   esperaConexao = new Thread(this);
   esperaConexao.start();
}

public void run()
{
   while(true)
   {
      try
      {
         // Passo 1: Criaum ServerSocket.
         server = new ServerSocket(3000,1);
         while(true)
         {
            // Passo 2: Espera uma conexão.
            display.append("Esperando por conexão\n");
            connection = server.accept();
            display.append("Conexão" + counter + "Recebido de: " +
               connection.getInetAddress().getHostName());

            // Passo 3: Obtém os fluxos de entrada e saída.
            dataOutputStream_App = new DataOutputStream(connection.getOutputStream());
            dataOutputStream_App.flush();
            dataInputStream_App = new DataInputStream(connection.getInputStream());

            // Passo 4: Processa conexão.
            String message = "SERVER>>> Conexão feita com sucesso";
            dataOutputStream_App.writeUTF(message);
            dataOutputStream_App.flush();
            PrimeiroPedido = true;
```

```
do
{
   try
   {
      message = dataInputStream_App.readUTF();
      StringTokenizer tokens = new StringTokenizer(message);
      Istatus = Integer.parseInt(tokens.nextToken());

      //Os dados são para o controlador = a numCont_B
      String numCont = tokens.nextToken();
      numCont_B = Byte.parseByte(numCont);

      //Tipo de referência (1 senoide) (2 quadrada) (4 triangular) (8 degrau)
      String ref1 = tokens.nextToken();
      ref1_B = Byte.parseByte(ref1);

      // Propriedades da referência do atuador 1
      float fre1 = Float.parseFloat(tokens.nextToken());
      fre1_S = Util.FreToPer(fre1); // Converte a frequencia na metade do periodo
      float amp1 = Float.parseFloat(tokens.nextToken());
      amp1_S = Util.trataTemp(amp1);// Converte o valor para short e multiplica por dois
      float ofs1 = Float.parseFloat(tokens.nextToken());
      ofs1_S = Util.trataTemp(ofs1);

      //Tipo de Controlador (0 PID) (1 Fuzzy)
      String cont = tokens.nextToken();
      tipoCont_B = Byte.parseByte(cont);

      //Propriedades do controlador 1
      float p1 = Float.parseFloat(tokens.nextToken());
      p1_S = Util.FloatShort(p1);
      float i1 = Float.parseFloat(tokens.nextToken());
      i1_S = Util.FloatShort(i1);
      float d1 = Float.parseFloat(tokens.nextToken());
      d1_S = Util.FloatShort(d1);

      if (PrimeiroPedido && Istatus==1)
      {
         //Já está estabelecido no programa do usuário (applet)
         // que ao fazer o "Start experiment" os primeiros parametros
```

```
                // pertencem ao controlador 1;
                if (numCont_B == Pic_1)
                {
                   sendProtocol(Pic_1,PICREC);
                }
                else
                {
                   sendProtocol(Pic_2,PICREC); //Na primeira vez em que enviar dados para o pic1 e pic2
                   PrimeiroPedido = false; // deve-se fazer o pedido de transmissão ao pic1
                   sendPic(Pic_1);        //A partir deste momento não é mais necessário pois
                                          //assim que os dados chegarem pela serial será feito
                                          // os novos pedidos (No método que trata os eventos
                                          //da serial.
                }
             }
             else
             {
                Bprotocol = true;  //Variável que abilita a transmissão dos dados para o pic
             }
          }
          catch (IOException cnfex)
          {
             display.append("Erro de I/O  ao ler dado do usuário remoto");
          }
       }
       while (!message.equals("0 0 0 0 0 0 0 0 0 0"));

       // Passo 5: Fecha a conexão.
       dataOutputStream_App.writeUTF("Termina_Conexao");
       dataOutputStream_App.flush();
       dataOutputStream_App.close();
       display.append( "\nConexão Terminada\n");
       dataInputStream_App.close();
       connection.close();
       ++counter;
    }
 }
 catch (EOFException eof)
 {
    display.append("Cliente terminou a conexão");
```

```
        }
        catch (IOException io)
        {
           display.append("Erro de I/O");
           io.printStackTrace();
        }
     }
} //Fim do construtor rodaServidor
} //Fim da classe interna rodaServidor
} //Fim da classe principal "sc"
```

Apêndice J: Código Java do programa Supervisório (*Applet)*

```
  ************************************Arquivo CApplet.java**********************************
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.*;
import javax.swing.JApplet.*;

public class CApplet extends JApplet  implements Runnable
{
   URL url;
   Socket client;
   DataOutputStream output;
   DataInputStream input;
   String message = "", Scont = "", Ref = "",Scont2 = "", Ref2 = "",StringIniciar="1";
   DefinePanel DPnivel,DPatuador;
   Maquete maqReal,maqDesenho;
   TituloPanel titulo;
   boolean showPlots[] = {true,true,true,true,true,true,true,true,true,true,true,true};
   public JTextField TFkp, TFki, TFkd, TFamplitude, TFfrequencia, TFoffset;
   JLabel Lkp,Lki,Lkd,Lamplitude,Lfrequencia,Loffset,Lmosgraf, Lcontrole1,Lcontrole2;
   String stringPainel[] = {"Tempo","Referência","Saída","Erro"};
   JCheckBox CBreferencia1,CBreferencia2,CBsala1,CBsala2,CBsala3,
         CBsala4,CBsala5,CBsala6,CBtempAmb1,CBtempAmb2,CBatuador1,CBatuador2;
   JPanel painel0,Pcontrolador1,Pcontrolador2,Preferencia1,Preferencia2,painel2, PsulEsq,
      p_v_completo,Pfuncoes,Pcontrole12,Pref12,Psala14,Psala25,Psala36,PtempAmb12,Patu12;
   JButton BenviarCon,BenviarCon2,Biniciar, Bparar,Bsair,BenviarRef,BenviarRef2;
   Thread thread1;
   private JRadioButton P, PI, RBfuzzy, PID, RBdegrau, RBsenoide, RBtriangular,
                  RBquadrada,RBarbitrario;
   private ButtonGroup grupo_selecao, BGreferencia;
   boolean execucao = false,enviar_1 = true;

   private Image logo1,logo2;

   public void init()
   {
//---------------------------------Painel_Controlador-------------------------
      logo1 = getImage(getDocumentBase(),"logunb.gif");
      logo2 = getImage(getDocumentBase(),"maquete.jpg");
      Pcontrolador1 = new JPanel();
      Pcontrolador1.setLayout(new GridLayout(1,6));
      RBclasse rbclasse = new RBclasse();

      P = new JRadioButton("Prop.",false);
      P.addItemListener(rbclasse);
      PI = new JRadioButton("Prop. Int.", false);
      PI.addItemListener(rbclasse);
      Pcontrolador1.add(P);
      Pcontrolador1.add(PI);

      TFkp = new JTextField();
      TFki = new JTextField();
      TFkd = new JTextField();
      Pcontrolador1.add(TFkp);
```

```
        Pcontrolador1.add(TFki);
        Pcontrolador1.add(TFkd);

        BenviarCon = new JButton("Send Controller 1");
        BenviarCon.addActionListener(rbclasse);
        Pcontrolador1.add(BenviarCon);

        Pcontrolador2 = new JPanel();
        Pcontrolador2.setLayout(new GridLayout(1,6));

        RBfuzzy = new JRadioButton("Fuzzy Control",false);
        PID = new JRadioButton("PID Control",true);
        RBfuzzy.addItemListener(rbclasse);
        PID.addItemListener(rbclasse);
        Pcontrolador2.add(PID);
        Pcontrolador2.add(RBfuzzy);

        Lkp = new JLabel("Kp");
        Lki = new JLabel("Ki");
        Lkd = new JLabel("Kd");
        Lkp.setHorizontalAlignment(SwingConstants.CENTER);
        Lki.setHorizontalAlignment(SwingConstants.CENTER);
        Lkd.setHorizontalAlignment(SwingConstants.CENTER);
        Pcontrolador2.add(Lkp);
        Pcontrolador2.add(Lki);
        Pcontrolador2.add(Lkd);
        BenviarCon2 = new JButton("Send Controller 2");
        BenviarCon2.addActionListener(rbclasse);
        BenviarCon2.setEnabled(false);
        Pcontrolador2.add(BenviarCon2);

        grupo_selecao = new ButtonGroup();
        grupo_selecao.add(P);
        grupo_selecao.add(PI);
        grupo_selecao.add(PID);
        grupo_selecao.add(RBfuzzy);

//--------------------------------Fim-Painel-Controlador----------------------

//--------------------------------Painel_Referencia 1 e 2--------------------------
        Preferencia1 = new JPanel();
        Preferencia1.setLayout(new GridLayout(1,6));
        RBdegrau = new JRadioButton("Step", false);
        RBquadrada = new JRadioButton("Square", true);
        RBdegrau.addItemListener(rbclasse);
        RBquadrada.addItemListener(rbclasse);
        Preferencia1.add(RBdegrau);
        Preferencia1.add(RBquadrada);
        TFfrequencia = new JTextField();
        TFamplitude = new JTextField();
        TFoffset = new JTextField();
        Preferencia1.add(TFfrequencia);
        Preferencia1.add(TFamplitude);
        Preferencia1.add(TFoffset);
        BenviarRef = new JButton("Send reference 1");
        BenviarRef.setEnabled(false);
        BenviarRef.addActionListener(rbclasse);
        Preferencia1.add(BenviarRef);

        Preferencia2 = new JPanel();
```

```
        Preferencia2.setLayout(new GridLayout(1,6));

        RBtriangular = new JRadioButton("Triangle",false);
        RBsenoide = new JRadioButton("Sine",false);
        RBtriangular.addItemListener(rbclasse);
        RBsenoide.addItemListener(rbclasse);
        Preferencia2.add(RBtriangular);
        Preferencia2.add(RBsenoide);

        BGreferencia = new ButtonGroup();
        BGreferencia.add(RBdegrau);
        BGreferencia.add(RBquadrada);
        BGreferencia.add(RBtriangular);
        BGreferencia.add(RBsenoide);
        BGreferencia.add(RBarbitrario);
        Loffset = new JLabel("Offset");
        Lamplitude = new JLabel("Amplitude");
        Lfrequencia = new JLabel("Frequency");
        Loffset.setHorizontalAlignment(SwingConstants.CENTER);
        Lamplitude.setHorizontalAlignment(SwingConstants.CENTER);
        Lfrequencia.setHorizontalAlignment(SwingConstants.CENTER);
        Preferencia2.add(Lfrequencia);
        Preferencia2.add(Lamplitude);
        Preferencia2.add(Loffset);
        BenviarRef2 = new JButton("Send reference 2");
        BenviarRef2.setEnabled(false);
        BenviarRef2.addActionListener(rbclasse);
        Preferencia2.add(BenviarRef2);

//---------------------------------Fim-Painel-Referencia----------------------

//---------------------------------Painel-Fun‡oes-----------------------------
        Pfuncoes = new JPanel();
        Pfuncoes.setLayout(new GridLayout(1,3));
        Biniciar = new JButton("Start Experiment");
        Biniciar.setEnabled(false);
        Biniciar.addActionListener(rbclasse);
        Pfuncoes.add(Biniciar);
        Bparar = new JButton("Stop Experiment");
        Bparar.setEnabled(false);
        Bparar.addActionListener(rbclasse);
        Pfuncoes.add(Bparar);
        Bsair = new JButton("Finish");
        Bsair.addActionListener(rbclasse);
        Pfuncoes.add(Bsair);
//-----------------------------------Fim-Painel-Funcoes-----------------------
//--------------------Painel lado esquerdo da regiao Sul-----------------------

        PsulEsq = new JPanel();
        PsulEsq.setLayout(new GridLayout(8,1));

        // Primeira linha do PsulEsq
        Lmosgraf = new JLabel("Show Plots:");
        Lmosgraf.setHorizontalAlignment(SwingConstants.CENTER);
        PsulEsq.add(Lmosgraf);

        // Segunda linha do PsulEsq
        Lcontrole1 = new JLabel("Controller-1");
        Lcontrole1.setHorizontalAlignment(SwingConstants.CENTER);
        Lcontrole2 = new JLabel("Controller-2");
```

```
Lcontrole2.setHorizontalAlignment(SwingConstants.CENTER);
Pcontrole12 = new JPanel();
Pcontrole12.setLayout(new GridLayout(1,2));
Pcontrole12.add(Lcontrole1);
Pcontrole12.add(Lcontrole2);
PsulEsq.add(Pcontrole12);

// Terceira linha do PsulEsq
CBreferencia1 = new JCheckBox("Reference-1",true);
CBreferencia1.setForeground(Color.red);
//CBreferencia1.setHorizontalAlignment(SwingConstants.CENTER);
CBreferencia1.addItemListener(rbclasse);
CBreferencia2 = new JCheckBox("Reference-2",true);
CBreferencia2.setForeground(Color.blue);
//CBreferencia2.setHorizontalAlignment(SwingConstants.CENTER);
CBreferencia2.addItemListener(rbclasse);
Pref12 = new JPanel();
Pref12.setLayout(new GridLayout(1,2));
Pref12.add(CBreferencia1);
Pref12.add(CBreferencia2);
PsulEsq.add(Pref12);

// Quarta linha do PsulEsq
CBsala1 = new JCheckBox("Sala-1",true);
CBsala1.setForeground(Color.cyan);
//CBsala1.setHorizontalAlignment(SwingConstants.CENTER);
CBsala1.addItemListener(rbclasse);
CBsala4 = new JCheckBox("Sala-3",true);
CBsala4.setForeground(Color.magenta);
//CBsala4.setHorizontalAlignment(SwingConstants.CENTER);
CBsala4.addItemListener(rbclasse);
Psala14 = new JPanel();
Psala14.setLayout(new GridLayout(1,2));
Psala14.add(CBsala1);
Psala14.add(CBsala4);
PsulEsq.add(Psala14);

// Quinta linha do PsulEsq

CBsala2 = new JCheckBox("Sala-2",true);
CBsala2.setForeground(Color.yellow);
//CBsala2.setHorizontalAlignment(SwingConstants.CENTER);
CBsala2.addItemListener(rbclasse);
CBsala5 = new JCheckBox("Sala-4",true);
CBsala5.setForeground(Color.orange);
//CBsala5.setHorizontalAlignment(SwingConstants.CENTER);
CBsala5.addItemListener(rbclasse);
Psala25 = new JPanel();
Psala25.setLayout(new GridLayout(1,2));
Psala25.add(CBsala2);
Psala25.add(CBsala5);
PsulEsq.add(Psala25);

// Sexta linha do PsulEsq
CBsala3 = new JCheckBox("",true);
CBsala3.setForeground(Color.lightGray);
//CBsala3.setHorizontalAlignment(SwingConstants.CENTER);
CBsala3.addItemListener(rbclasse);
CBsala6 = new JCheckBox("Sala-5",true);
CBsala6.setForeground(Color.pink);
```

```
        //CBsala6.setHorizontalAlignment(SwingConstants.CENTER);
        CBsala6.addItemListener(rbclasse);
        Psala36 = new JPanel();
        Psala36.setLayout(new GridLayout(1,2));
        Psala36.add(CBsala3);
        Psala36.add(CBsala6);
        PsulEsq.add(Psala36);

        // Sétima linha do PsulEsq
        CBtempAmb1 = new JCheckBox("TempAmb-1",true);
        CBtempAmb1.setForeground(Color.green);
        //CBtempAmb1.setHorizontalAlignment(SwingConstants.CENTER);
        CBtempAmb1.addItemListener(rbclasse);
        CBtempAmb2 = new JCheckBox("TempAmb-2",true);
        CBtempAmb2.setForeground(Color.white);
        //CBtempAmb2.setHorizontalAlignment(SwingConstants.CENTER);
        CBtempAmb2.addItemListener(rbclasse);
        PtempAmb12 = new JPanel();
        PtempAmb12.setLayout(new GridLayout(1,2));
        PtempAmb12.add(CBtempAmb1);
        PtempAmb12.add(CBtempAmb2);
        PsulEsq.add(PtempAmb12);

        // Oitava linha do PsulEsq
        CBatuador1 = new JCheckBox("Actuator-1",true);
        CBatuador1.setForeground(Color.red);
        //CBatuador1.setHorizontalAlignment(SwingConstants.CENTER);
        CBatuador1.addItemListener(rbclasse);
        CBatuador2 = new JCheckBox("Actuator-2",true);
        CBatuador2.setForeground(Color.yellow);
        //CBatuador2.setHorizontalAlignment(SwingConstants.CENTER);
        CBatuador2.addItemListener(rbclasse);
        Patu12 = new JPanel();
        Patu12.setLayout(new GridLayout(1,2));
        Patu12.add(CBatuador1);
        Patu12.add(CBatuador2);
        PsulEsq.add(Patu12);

//-----------------------------Fim-painel-Sul esquerdo--------------------------

        Container c = getContentPane();
        titulo = new TituloPanel(logo1);
        JPanel Pnorte = new JPanel();
        Pnorte.setLayout(new BorderLayout());
        Pnorte.add(titulo,BorderLayout.NORTH);
        painel0 = new JPanel();
        painel0.setLayout(new GridLayout(7,1));
        painel0.add(new JLabel("Define Controller"));
        painel0.add(Pcontrolador1);
        painel0.add(Pcontrolador2);
        painel0.add(new JLabel("Define reference"));
        painel0.add(Preferencia1);
        painel0.add(Preferencia2);
        painel0.add(Pfuncoes);
        Pnorte.add(painel0,BorderLayout.SOUTH);
        c.add(Pnorte,BorderLayout.NORTH);
        p_v_completo = new JPanel();
        p_v_completo.setLayout(new GridLayout(1,3));
        p_v_completo.add(PsulEsq);
        maqReal = new Maquete(logo2);
```

```
        maqDesenho = new Maquete();
        p_v_completo.add(maqReal);
        p_v_completo.add(maqDesenho);
        c.add(p_v_completo,BorderLayout.SOUTH);

        DPnivel = new DefinePanel(true);
        DPnivel.setBackground(Color.black);
        DPatuador = new DefinePanel(false);
        DPatuador.setBackground(Color.black);
        painel2 = new JPanel();
        painel2.setLayout(new GridLayout(2,1));
        painel2.add(DPnivel);
        painel2.add(DPatuador);
        c.add(painel2,BorderLayout.CENTER);

        setSize(820,980);
    }

    public void start()
    {
        int w;
        try
        {
            //Passo 1: Cria um Socket para fazer a conexão.
            client = new Socket(InetAddress.getByName("164.41.49.94"), 3000);

            //Passo 2: Obtém os fluxos de entrada e saída.
            output = new DataOutputStream(client.getOutputStream());
            output.flush();
            input = new DataInputStream(client.getInputStream());
        }
        catch (IOException io)
        {
            JOptionPane.showMessageDialog(null,"Erro:002! Atualize a página.",
             "ERRO!", JOptionPane.ERROR_MESSAGE);
            io.printStackTrace();
        }

        thread1 = new Thread(this,"thread1");
        thread1.start();
    }
    public void destroy()
    {
        if( enviar_1)
            sendData("0 0 0 0 0 0 0 0 0 0");
//      try
//      {
//          input.close();
//          output.close();
//          client.close();
//      }
//      catch (IOException io)
//      {
//          //JOptionPane.showMessageDialog(null,"Houve um erro ao fechar a conexão!",
//          //"ERRO!", JOptionPane.ERROR_MESSAGE);
//          //io.printStackTrace();
//      }
    }
    public void stop()
    {
```

```
      if( enviar_1)
      {
         sendData("0 0 0 0 0 0 0 0 0 0");
         enviar_1=false;
      }
//try
//{
//input.close();
//output.close();
//client.close();
//}
///catch (IOException io) {
//JOptionPane.showMessageDialog(null,"Houve um erro ao fechar a conexão!",
//"ERRO!",  JOptionPane.ERROR_MESSAGE);
//io.printStackTrace();
//}
   }

   public void run()
   {
      try
      {
         //Passo 3: Processa conexão.
         message = input.readUTF();
         do
         {
            message = input.readUTF();
            if(!message.equals("Termina_Conexao"))
            {
               DPnivel.desenha2(message);
               DPatuador.desenha2(message);
               maqDesenho.desenha(message);
            }
         }
         while (!message.equals("Termina_Conexao"));

         //Passo 4: Fecha a conexão.
         input.close();
         output.close();
         client.close();
      }
      catch (EOFException eof)
      {
         JOptionPane.showMessageDialog(null,"Servidor terminou a conexão!",
         "ATENÇÃO !", JOptionPane.INFORMATION_MESSAGE);
      }
      catch (IOException io)
      {
         JOptionPane.showMessageDialog(null,"Erro:003! Atualize a página.",
         "ERRO!", JOptionPane.ERROR_MESSAGE);
         io.printStackTrace();
      }
   }
   private void sendData(String s)
   {
      try
      {
         output.writeUTF( s );
         output.flush();
      }
```

```
        catch (IOException cnfex)
        {
        JOptionPane.showMessageDialog(null,"Erro:004! Atualize a página.",
           "ERRO!", JOptionPane.ERROR_MESSAGE);
        }
    }

private class RBclasse implements ItemListener, ActionListener
{
    public void itemStateChanged( ItemEvent e)
    {
        if( e.getSource() == P)
        {
           TFkp.setEditable(true);
           TFki.setEditable(false);
           TFki.setText("");
           TFkd.setEditable(false);
           TFkd.setText("");
        }
        if( e.getSource() == PI)
        {
           TFkp.setEditable(true);
           TFkd.setEditable(false);
           TFkd.setText("");
           TFki.setEditable(true);
        }
        if( e.getSource() == RBfuzzy)
        {
           TFki.setEditable(false);
           TFki.setText("");
           TFkd.setEditable(false);
           TFkd.setText("");
           TFkp.setEditable(false);
           TFkp.setText("");
        }
        if( e.getSource() == PID)
        {
           TFki.setEditable(true);
           TFkp.setEditable(true);
           TFkd.setEditable(true);
        }
        if( e.getSource() == RBsenoide)
        {
           TFoffset.setEditable(true);
           TFamplitude.setEditable(true);
           TFfrequencia.setEditable(true);
        }
        if( e.getSource() == RBquadrada)
        {
           TFamplitude.setEditable(true);
           TFoffset.setEditable(true);
           TFfrequencia.setEditable(true);
        }
        if( e.getSource() == RBdegrau)
        {
           TFamplitude.setEditable(true);
           TFoffset.setText("");
           TFfrequencia.setText("");
           TFoffset.setEditable(false);
           TFfrequencia.setEditable(false);
```

```
        }
        if( e.getSource() == RBtriangular)
        {
           TFamplitude.setEditable(true);
           TFoffset.setEditable(true);
           TFfrequencia.setEditable(true);
        }

//-----------Inicio do bloco que configura quais curvas serão plotadas----------

        if( e.getSource() == CBreferencia1)
        {
           if( e.getStateChange() == ItemEvent.SELECTED)
           {
              showPlots[0] =true;
              DPnivel.graficos(showPlots);
           }
           else
           {
              showPlots[0] =false;
              DPnivel.graficos(showPlots);
           }
        }
        if( e.getSource() == CBsala1)
        {
           if( e.getStateChange() == ItemEvent.SELECTED)
           {
              showPlots[1] =true;
              DPnivel.graficos(showPlots);
           }
           else
           {
              showPlots[1] =false;
              DPnivel.graficos(showPlots);
           }
        }
        if( e.getSource() == CBsala2)
        {
           if( e.getStateChange() == ItemEvent.SELECTED)
           {
              showPlots[2] =true;
              DPnivel.graficos(showPlots);
           }
           else
           {
              showPlots[2] =false;
              DPnivel.graficos(showPlots);
           }
        }
//      if( e.getSource() == CBsala3)
//      {
//         if( e.getStateChange() == ItemEvent.SELECTED)
//         {
//            showPlots[3] =true;
//            DPnivel.graficos(showPlots);
//         }
//         else
//         {
//            showPlots[3] =false;
//            DPnivel.graficos(showPlots);
```

```
//         }
//      }
        if( e.getSource() == CBtempAmb1)
        {
           if( e.getStateChange() == ItemEvent.SELECTED)
           {
              showPlots[4] =true;
              DPnivel.graficos(showPlots);
           }
           else
           {
              showPlots[4] =false;
              DPnivel.graficos(showPlots);
           }
        }
        if( e.getSource() == CBatuador1)
        {
           if( e.getStateChange() == ItemEvent.SELECTED)
           {
              showPlots[5] =true;
              DPatuador.graficos(showPlots);
           }
           else
           {
              showPlots[5] =false;
              DPatuador.graficos(showPlots);
           }
        }
        if( e.getSource() == CBreferencia2)
        {
           if( e.getStateChange() == ItemEvent.SELECTED)
           {
              showPlots[6] =true;
              DPnivel.graficos(showPlots);
           }
           else
           {
              showPlots[6] =false;
              DPnivel.graficos(showPlots);
           }
        }
        if( e.getSource() == CBsala4)
        {
           if( e.getStateChange() == ItemEvent.SELECTED)
           {
              showPlots[7] =true;
              DPnivel.graficos(showPlots);
           }
           else
           {
              showPlots[7] =false;
              DPnivel.graficos(showPlots);
           }
        }
        if( e.getSource() == CBsala5)
        {
           if( e.getStateChange() == ItemEvent.SELECTED)
           {
              showPlots[8] =true;
              DPnivel.graficos(showPlots);
```

```
            }
            else
            {
               showPlots[8] =false;
               DPnivel.graficos(showPlots);
            }
         }
         if( e.getSource() == CBsala6)
         {
            if( e.getStateChange() == ItemEvent.SELECTED)
            {
               showPlots[9] =true;
               DPnivel.graficos(showPlots);
            }
            else
            {
               showPlots[9] =false;
               DPnivel.graficos(showPlots);
            }
         }
         if( e.getSource() == CBtempAmb2)
         {
            if( e.getStateChange() == ItemEvent.SELECTED)
            {
               showPlots[10] =true;
               DPnivel.graficos(showPlots);
            }
            else
            {
               showPlots[10] =false;
               DPnivel.graficos(showPlots);
            }
         }
         if( e.getSource() == CBatuador2)
         {
            if( e.getStateChange() == ItemEvent.SELECTED)
            {
               showPlots[11] =true;
               DPatuador.graficos(showPlots);
            }
            else
            {
               showPlots[11] =false;
               DPatuador.graficos(showPlots);
            }
         }
      }
//-----------Fim do bloco que configura quais curvas serão plotadas----------

      public void actionPerformed( ActionEvent e)
      {
         if( e.getSource() == Biniciar)
         {
            execucao = true;
            sendData(StringIniciar + " 1 " + Ref + " " + Scont);
            sendData(StringIniciar + " 2 " + Ref2 + " " + Scont2);
            Biniciar.setEnabled(false);
            Bparar.setEnabled(true);
         }
         if( e.getSource() == Bparar)
```

```
        {
            sendData("0 0 0 0 0 0 0 0 0 0");
//          StringIniciar = "2";
            enviar_1=false;
            execucao = false;
            BenviarRef.setEnabled(false);
            BenviarRef2.setEnabled(false);
            BenviarCon2.setEnabled(false);
            BenviarCon.setEnabled(false);
        }

        if( e.getSource() == Bsair)
        {
            try
            {
                url = new URL("http://164.41.49.94/termico/dados.jsp");
                AppletContext browser = getAppletContext();
                enviar_1=false;
                destroy();
                browser.showDocument(url);
            }
            catch(MalformedURLException u)
            {
            }
        }

        if( e.getSource() == BenviarCon)
        {
            String Skp = TFkp.getText();
            String Skd = TFkd.getText();
            String Ski = TFki.getText();
            if(P.isSelected())
                if(!Util.campoVazio(Skp,"Kp") && Util.campoCorreto(Skp,"Kp"))
                {
                Scont ="1 " + Skp + " 0 0"; // O primeiro um corresponde ao tipo de controlador que é o PID
                BenviarCon2.setEnabled(true);
                }
            if(RBfuzzy.isSelected())
            {
                Scont = "2 0 0 0"; // O número "2" corresponde ao tipo de controlador que é o fuzzy.
                BenviarCon2.setEnabled(true);
            }
            if(PID.isSelected())
                if(!Util.campoVazio(Skp,"Kp") && Util.campoCorreto(Skp,"Kp"))
                    if(!Util.campoVazio(Skd,"Kd") && Util.campoCorreto(Skd,"Kd"))
                        if(!Util.campoVazio(Ski,"Ki") && Util.campoCorreto(Ski,"Ki"))
                        {
                            Scont ="1 " + Skp + " " + Ski + " " + Skd;
                            BenviarCon2.setEnabled(true);
                        }
            if(PI.isSelected())
                if(!Util.campoVazio(Skp,"Kp") && Util.campoCorreto(Skp,"Kp"))
                    if(!Util.campoVazio(Ski,"Ki") && Util.campoCorreto(Ski,"Ki"))
                    {
                        Scont ="1 " + Skp + " " + Ski + " 0";
                        BenviarCon2.setEnabled(true);
                    }
            if(execucao)
                sendData(StringIniciar + " 1 " + Ref + " " + Scont); // Envia o "Status"; Para qual controlador "1";
        }                                                   // A referência; e o controlador;
```

```
if( e.getSource() == BenviarRef)
{
   String Samp = TFamplitude.getText();
   String Sfre = TFfrequencia.getText();
   String Soff = TFoffset.getText();
   if(RBdegrau.isSelected())
      if(!Util.campoVazio(Samp,"Amplitude") &&    Util.campoCorreto(Samp,"Amplitude"))
      {
         Ref = "8 0 " + Samp + " 0"; // tipo de referencia: "8" = Degrau
         BenviarRef2.setEnabled(true);
   }
   if(RBsenoide.isSelected())
      if(!Util.campoVazio(Sfre,"Frequencia") && Util.campoCorreto(Sfre,"Frequencia"))
         if(!Util.campoVazio(Samp,"Amplitude") && Util.campoCorreto(Samp,"Amplitude"))
            if(!Util.campoVazio(Soff,"Offset") && Util.campoCorreto(Soff,"Offset"))
            {
               Ref = "1 " + Sfre + " " + Samp + " " + Soff;// tipo de referencia: "1" = Senoide
               BenviarRef2.setEnabled(true);
               }
   if(RBquadrada.isSelected())
      if(!Util.campoVazio(Sfre,"Frequencia") && Util.campoCorreto(Sfre,"Frequencia"))
         if(!Util.campoVazio(Samp,"Amplitude") && Util.campoCorreto(Samp,"Amplitude"))
            if(!Util.campoVazio(Soff,"Offset") && Util.campoCorreto(Soff,"Offset"))
            {
               Ref = "2 " + Sfre + " " + Samp + " " + Soff;// tipo de referencia: "2" = Quadrada
               BenviarRef2.setEnabled(true);
            }
   if(RBtriangular.isSelected())
      if(!Util.campoVazio(Sfre,"Frequencia") && Util.campoCorreto(Sfre,"Frequencia"))
         if(!Util.campoVazio(Samp,"Amplitude") && Util.campoCorreto(Samp,"Amplitude"))
            if(!Util.campoVazio(Soff,"Offset") && Util.campoCorreto(Soff,"Offset"))
            {
               Ref = "4 " + Sfre + " " + Samp + " " + Soff;// tipo de referencia: "4" = Triangular
               BenviarRef2.setEnabled(true);
            }
   if(execucao) //Se o experimento está em andamento
   {
      sendData(StringIniciar + " 1 " + Ref + " " + Scont);
   }
}
if( e.getSource() == BenviarCon2)
{
   String Skp2 = TFkp.getText();
   String Skd2 = TFkd.getText();
   String Ski2 = TFki.getText();
   if(P.isSelected())
      if(!Util.campoVazio(Skp2,"Kp") && Util.campoCorreto(Skp2,"Kp"))
      {
         Scont2 ="1 " + Skp2 + " 0 0";
         BenviarRef.setEnabled(true);
      }
   if(RBfuzzy.isSelected())
   {
      Scont2 = "2 0 0 0";
      BenviarRef.setEnabled(true);
   }
   if(PID.isSelected())
      if(!Util.campoVazio(Skp2,"Kp") && Util.campoCorreto(Skp2,"Kp"))
         if(!Util.campoVazio(Skd2,"Kd") && Util.campoCorreto(Skd2,"Kd"))
            if(!Util.campoVazio(Ski2,"Ki") && Util.campoCorreto(Ski2,"Ki"))
```

```
                {
                   Scont2 ="1 " + Skp2 + " " + Ski2 + " " + Skd2;
                   BenviarRef.setEnabled(true);
                }
          if(PI.isSelected())
             if(!Util.campoVazio(Skp2,"Kp") && Util.campoCorreto(Skp2,"Kp"))
                if(!Util.campoVazio(Ski2,"Ki") && Util.campoCorreto(Ski2,"Ki"))
                {
                   Scont2 ="1 " + Skp2 + " " + Ski2 + " 0";
                   BenviarRef.setEnabled(true);
                }
          if(execucao)
             sendData(StringIniciar + " 2 " + Ref2 + " " + Scont2);
       }
       if( e.getSource() == BenviarRef2)
       {
          String Samp2 = TFamplitude.getText();
          String Sfre2 = TFfrequencia.getText();
          String Soff2 = TFoffset.getText();

          if(RBdegrau.isSelected())
             if(!Util.campoVazio(Samp2,"Amplitude") && Util.campoCorreto(Samp2,"Amplitude"))
             {
                Ref2 = "8 0 " + Samp2 + " 0";
                Biniciar.setEnabled(true);
             }
          if(RBsenoide.isSelected())
             if(!Util.campoVazio(Sfre2,"Frequencia") && Util.campoCorreto(Sfre2,"Frequencia"))
                if(!Util.campoVazio(Samp2,"Amplitude") && Util.campoCorreto(Samp2,"Amplitude"))
                   if(!Util.campoVazio(Soff2,"Offset") && Util.campoCorreto(Soff2,"Offset"))
                   {
                      Ref2 = "1 " + Sfre2 + " " + Samp2 + " " + Soff2;
                      Biniciar.setEnabled(true);
                   }
          if(RBquadrada.isSelected())
             if(!Util.campoVazio(Sfre2,"Frequencia") && Util.campoCorreto(Sfre2,"Frequencia"))
                if(!Util.campoVazio(Samp2,"Amplitude") && Util.campoCorreto(Samp2,"Amplitude"))
                   if(!Util.campoVazio(Soff2,"Offset") && Util.campoCorreto(Soff2,"Offset"))
                   {
                      Ref2 = "2 " + Sfre2 + " " + Samp2 + " " + Soff2;
                      Biniciar.setEnabled(true);
                   }
          if(RBtriangular.isSelected())
             if(!Util.campoVazio(Sfre2,"Frequencia") && Util.campoCorreto(Sfre2,"Frequencia"))
                if(!Util.campoVazio(Samp2,"Amplitude")  && Util.campoCorreto(Samp2,"Amplitude"))
                   if(!Util.campoVazio(Soff2,"Offset") && Util.campoCorreto(Soff2,"Offset"))
                   {
                      Ref2 = "4 " + Sfre2 + " " + Samp2 + " " + Soff2;
                      Biniciar.setEnabled(true);
                   }
          if(execucao)
          {
             sendData(StringIniciar + " 2 " + Ref2 + " " + Scont2);
             Biniciar.setEnabled(false);
          }
       }
    }//---------------------------- Fim dos eventos de ação "actionPerformed"-------------------------------------
}// Fim da classe interna rbClasse
} // Fim da classe principal
```

**********Arquivo DefinePanel.java**********

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.text.*;
import java.util.*;

public class DefinePanel extends JPanel
{
   private int v = 0, u = 0;
   private final byte Pic_1 = 1;
   private final byte Pic_2 = 2;
   private int t_Pic1[] = new int[1200];
   private int ref_Pic1[] = new int[1200];
   private int s1_Pic1[] = new int[1200];
   private int s2_Pic1[] = new int[1200];
   private int s3_Pic1[] = new int[1200];
   private int ta_Pic1[] = new int[1200];
   private int a_Pic1[] = new int[1200];
   private int t_Pic2[] = new int[1200];
   private int ref_Pic2[] = new int[1200];
   private int s1_Pic2[] = new int[1200];
   private int s2_Pic2[] = new int[1200];
   private int s3_Pic2[] = new int[1200];
   private int ta_Pic2[] = new int[1200];
   private int a_Pic2[] = new int[1200];

   private boolean permissao = false, seg_tela = false,
           v_zerado1 = false, painel1, finished=false,
           BplotaPic_1,BplotaPic_2;

   private boolean showPlots[] = {true,true,true,true,true,true,true,true,true,true,true,true};

public DefinePanel(boolean p1)
{
   painel1 = p1;
}

public void paintComponent(Graphics g)
{
   super.paintComponent(g);
   if(painel1)
   {
      g.setColor(Color.white);
      g.drawString("Temp",2,10);
      g.drawString("00",42,174);
      g.drawString("20",42,142);
      g.drawString("40",42,111);
      g.drawString("60",42,79);
      g.drawString("80",42,47);
      g.drawString("100",36,15);
   }
   else
   {
      g.setColor(Color.white);
      g.drawString("Volts",2,10);
      g.drawString("000",36,174);
      g.drawString("040",36,142);
      g.drawString("080",36,111);
```

```
    g.drawString("120",36,79);
    g.drawString("160",36,47);
    g.drawString("200",36,15);
}

for(int i=1; i<=20;i++)
{
    g.drawLine(60+34*i,168,60+34*i,172); //PONTOS EIXO  X
}

for(int i=0; i<10;i++)
{
    g.drawLine(58,10+16*i,62,10+16*i);  // Pontos eixo Y
}

g.drawLine(58,170,740,170); //Eixo x
g.drawLine(60,10,60,172);   //EIXO y
g.drawString("Seconds",380,200);

if((!seg_tela))
{
    g.drawString("20",87,186);
    g.drawString("40",121,186);
    g.drawString("60",155,186);
    g.drawString("80",189,186);
    g.drawString("100",220,186);
    g.drawString("120",254,186);
    g.drawString("140",288,186);
    g.drawString("160",322,186);
    g.drawString("180",356,186);
    g.drawString("200",390,186);
    g.drawString("220",424,186);
    g.drawString("240",458,186);
    g.drawString("260",492,186);
    g.drawString("280",526,186);
    g.drawString("300",560,186);
    g.drawString("320",594,186);
    g.drawString("340",628,186);
    g.drawString("360",662,186);
    g.drawString("380",696,186);
    g.drawString("400",730,186);
}
else
{
    g.drawString("420",84,186);
    g.drawString("440",118,186);
    g.drawString("460",152,186);
    g.drawString("480",186,186);
    g.drawString("500",220,186);
    g.drawString("520",254,186);
    g.drawString("540",288,186);
    g.drawString("560",322,186);
    g.drawString("580",356,186);
    g.drawString("600",390,186);
    g.drawString("620",424,186);
    g.drawString("640",458,186);
    g.drawString("660",492,186);
    g.drawString("680",526,186);
    g.drawString("700",560,186);
    g.drawString("720",594,186);
```

```
    g.drawString("740",628,186);
    g.drawString("760",662,186);
    g.drawString("780",696,186);
    g.drawString("800",730,186);
  }

  if ((permissao) && (painel1))
  {
    if (showPlots[0])
    {
      g.setColor(Color.red);
      g.drawPolyline(t_Pic1,ref_Pic1,v);
    }
    if (showPlots[1])
    {
      g.setColor(Color.cyan);
      g.drawPolyline(t_Pic1,s1_Pic1,v);
    }
    if (showPlots[2])
    {
      g.setColor(Color.yellow);
      g.drawPolyline(t_Pic1,s2_Pic1,v);
    }
    if (showPlots[3]) //Sala 3 do pic1 não exite
    {
      g.setColor(Color.lightGray);
      g.drawPolyline(t_Pic1,s3_Pic1,v);
    }
    if (showPlots[4])
    {
      g.setColor(Color.green);
      g.drawPolyline(t_Pic1,ta_Pic1,v);
    }
    if (showPlots[6])   //ATENÇÃO: O ShowPlots[5 e 11] são atuadores!
    {
      g.setColor(Color.blue);
      g.drawPolyline(t_Pic2,ref_Pic2,u);
    }
    if (showPlots[7])
    {
      g.setColor(Color.magenta);
      g.drawPolyline(t_Pic2,s1_Pic2,u);
    }
    if (showPlots[8])
    {
      g.setColor(Color.orange);
      g.drawPolyline(t_Pic2,s2_Pic2,u);
    }
    if (showPlots[9])
    {
      g.setColor(Color.pink);
      g.drawPolyline(t_Pic2,s3_Pic2,u);
    }
    if (showPlots[10])
    {
      g.setColor(Color.white);
      g.drawPolyline(t_Pic2,ta_Pic2,u);
    }
  }
  else
```

```
   {
      if ((permissao) && (!painel1))
      {
         if (showPlots[5])
         {
            g.setColor(Color.red);
            g.drawPolyline(t_Pic1,a_Pic1,v);
         }
         if (showPlots[11])
         {
            g.setColor(Color.yellow);
            g.drawPolyline(t_Pic2,a_Pic2,u);
         }
      }
   }
}
public void graficos(boolean show[])
{
   showPlots = show;
   this.repaint();
}
public void desenha2(String s)
{
   try
   {
      permissao = true;
      if(!s.equals("NOVO GRAFICO"))
      {
         StringTokenizer tokens = new StringTokenizer(s);

         byte numPic = Byte.parseByte(tokens.nextToken());
         float tempo = Float.parseFloat(tokens.nextToken());
         float ref = Float.parseFloat(tokens.nextToken());
         float sala1 = Float.parseFloat(tokens.nextToken());
         float sala2 = Float.parseFloat(tokens.nextToken());
         float sala3 = Float.parseFloat(tokens.nextToken());
         float tamb = Float.parseFloat(tokens.nextToken());
         float atu = Float.parseFloat(tokens.nextToken());

         if ((tempo >=799) && (!finished) && (painel1))
         {
         finished = true;
         JOptionPane.showMessageDialog(null,"Your Experiment finished. Click Button  \"Finish\" ",
         "Finished Experiment !",JOptionPane.INFORMATION_MESSAGE);
         }

         if ((tempo<400) && (seg_tela)) //Caso o tempo de um dos Pics chegue aos 400 s (segunda tela) e o outro
         {                       // posteriormente não, então o outro (o atrasado) só plotará quando seu
                                 // seu tempo chegar aos 400 segundos.
            if (numPic == Pic_2)
            {
               BplotaPic_2 = false;
            }
            else
            {
               BplotaPic_1 = false;
            }
         }
         else
         {
```

```
            BplotaPic_2 = true;
            BplotaPic_1 = true;
        }
        if (tempo >= 400) //if numero
        {
            tempo -= 400;
            seg_tela = true; // Habilita a segunda tela
        }

        if ((seg_tela) && (!v_zerado1)) // Se está na segunda tela e o v ainda não foi zerado então zera-o.
        {                       // Vale ressaltar que esta definição assegura que o v será zerado
                                // apenas uma vez. Ele não pode ser zerado no if numero 1 pois
                                // após 400 segundos aquela definição sempre será verdadeira
            v = 0;
            u = 0;
            v_zerado1 = true;
        }

        if(numPic == Pic_1 && BplotaPic_1)
        {
            t_Pic1[v] = (int) (60 + (1.7*tempo));
            ref_Pic1[v] = (int) (170 -(8*ref/5));
            s1_Pic1[v] = (int) (170 -(8*sala1/5));
            s2_Pic1[v] = (int) (170 -(8*sala2/5));
            s3_Pic1[v] = (int) (170 -(8*sala3/5));
            ta_Pic1[v] = (int) (170 -(8*tamb/5));
            a_Pic1[v] = (int) (170 -(16*atu));
            v += 1;
        }
        if(numPic == Pic_2 && BplotaPic_2)
        {
            t_Pic2[u] = (int) (60 + (1.7*tempo));
            ref_Pic2[u] = (int) (170 -(8*ref/5));
            s1_Pic2[u] = (int) (170 -(8*sala1/5));
            s2_Pic2[u] = (int) (170 -(8*sala2/5));
            s3_Pic2[u] = (int) (170 -(8*sala3/5));
            ta_Pic2[u] = (int) (170 -(8*tamb/5));
            a_Pic2[u] = (int) (170 -(16*atu));
            u += 1;
        }
        this.repaint();
    }
    else // Caso seja novo gráfico zera os vetores.
    {   //Isto ocorre quando o usuário pressiona "Stop" e depois reinicia o esperimento
        for(int i=0; i<v;i++)
        {
            t_Pic1[i]=0;
            ref_Pic1[i]=0;
            s1_Pic1[i]=0;
            s2_Pic1[i]=0;
            s3_Pic1[i]=0;
            ta_Pic1[i]=0;
            a_Pic1[i]=0;
        }
        for(int i=0; i<u;i++)
        {
            t_Pic2[i]=0;
            ref_Pic2[i]=0;
            s1_Pic2[i]=0;
            s2_Pic2[i]=0;
```

```
                s3_Pic2[i]=0;
                ta_Pic2[i]=0;
                a_Pic2[i]=0;
            }
            v = 0;
            u = 0;
            seg_tela = false;
            v_zerado1 = false;
        }
    }
    catch(NumberFormatException nfe)
    {
        JOptionPane.showMessageDialog(null, "Dado com Erro !", "Dado com Erro !",
            JOptionPane.ERROR_MESSAGE);
    }
}
}
```

**********Arquivo Maquete.java**********

```
import java.awt.*;
import javax.swing.*;
import java.util.*;
import java.text.DecimalFormat;

public class Maquete extends JPanel
{
   public Image logo2;
   private boolean desenho;
   String Ssala1="25.0",Ssala2="25.0",Ssala3="25.0",StempAmb1="25.0",Ssala4="25.0",Ssala5="25.0",
         Ssala6="",StempAmb2="25.0";
   float Isala1=25,Isala2=25,Isala3=25,Isala4=25,Isala5=25,Isala6=25,ItempAmb1=25,ItempAmb2=25;
   final int Pic_1 = 1,Pic_2 = 2;
   int Bnum;
   DecimalFormat formata;

   public Maquete(Image image)
   {
      logo2 = image;
      desenho = false;
   }
   public Maquete()
   {
      desenho = true;
   //  formata = new DecimalFormat("00.00");
   }
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      if (!desenho)
      {
         g.drawImage(logo2,0,0,getWidth(),getHeight(),this);
      }
      else
      {
         int W = -10 + getWidth() ;
         int H = getHeight() - 10;
         int c1x = 5 + (int) Math.floor(0.15*W);
         int c1y = 5 + (int) Math.floor(0.2*H);
         int c2x =  c1x + (int) Math.floor(0.3*W);
         int c2y =  c1y + (int)Math.floor(0.23*H);
         int c3y =  c1y + (int)Math.floor(0.3*H);
         int c4x =  c2x + (int)Math.floor(0.2*W);
         g.drawRect(5,5,W,H);
         g.drawRect(c1x,c1y,c2x-c1x,c2y-c1y); //Sala 2
         g.drawRect(c1x,c2y,c2x-c1x,H-c1y-c2y+10);//Sala 1
         g.drawRect(c2x,c1y,c4x-c2x,c3y-c1y);//Sala 3
         g.drawRect(c4x,c1y,c4x-c2x,c3y-c1y);//Sala 4
         g.drawRect(c2x,c3y,2*(c4x-c2x),H-c1y-c3y+10); //Sala 5
         g.drawLine(5,H-c1y+10,c1x,H-c1y+10);
         g.drawLine(c4x,c1y,W+5,c1y);

//Segurança para que os parametros das cores não esteja fora dos limites
         if (Isala1>100)
         {
            Isala1 = 100;
         }
         if (Isala2>100)
```

```
            {
               Isala2 = 100;
            }
            if (Isala3>100)
            {
               Isala3 = 100;
            }
            if (Isala4>100)
            {
               Isala4 = 100;
            }
            if (Isala5>100)
            {
               Isala5 = 100;
            }
            if (ItempAmb1>100)
            {
               ItempAmb1 = 100;
            }
            if (ItempAmb2>100)
            {
               ItempAmb2 = 100;
            }
//---------------------------------------------------------------------
            g.setColor(new Color(255,(int)(250-2.5*Isala2),0));
            g.fillRect(c1x+2,c1y+2,c2x-c1x-2,c2y-c1y-2); //Sala 2
            g.setColor(new Color(255,(int)(250-2.5*Isala1),0));
            g.fillRect(c1x+2,c2y+2,c2x-c1x-2,H-c1y-c2y+10-2);//Sala 1
            g.setColor(new Color(255,(int)(250-2.5*Isala3),0));
            g.fillRect(c2x+2,c1y+2,c4x-c2x-2,c3y-c1y-2);//Sala 3
            g.setColor(new Color(255,(int)(250-2.5*Isala4),0));
            g.fillRect(c4x+2,c1y+2,c4x-c2x-2,c3y-c1y-2);//Sala 4
            g.setColor(new Color(255,(int)(250-2.5*Isala5),0));
            g.fillRect(c2x+2,c1y+c3y-c1y+2,2*(c4x-c2x-1),H-c1y-c3y+10-2); //Sala 5
            g.setColor(new Color(255,(int)(250-2.5*ItempAmb2),0));
            g.fillRect(7,H-c1y+12,W-3,c1y-7); //Sala ambiente inferior
            g.fillRect(c4x+c4x-c2x+2,c1y+2,W-(c4x+c4x-c2x)+2,H-(c1y+7));
            g.setColor(new Color(255,(int)(250-2.5*ItempAmb1),0));
            g.fillRect(7,7,W-3,c1y-8); // Sala ambiente superior
            g.fillRect(7,7,c1x-7,H-c1y+3);

            g.setColor(Color.black);
            g.setFont(new Font("Serif",Font.PLAIN,12));
            g.drawString("Sala-2",((c1x+c2x)/2)-15,((c1x+c2x)/2)-20);
            g.drawString(Ssala2+"ºC",((c1x+c2x)/2)-20,((c1x+c2x)/2)-8);
            g.drawString("Sala-1",((c1x+c2x)/2)-13,((H-c1y+c2y+10)/2)-5);
            g.drawString(Ssala1+"ºC",((c1x+c2x)/2)-17,((H-c1y+c2y+10)/2)+7);
            g.drawString("Sala-3",((c4x+c2x)/2)-13,((c1y+c3y)/2)-5);
            g.drawString(Ssala3+"ºC",((c4x+c2x)/2)-17,((c1y+c3y)/2)+7);
            g.drawString("Sala-4",((3*c4x-c2x)/2)-13,((c1y+c3y)/2)-5);
            g.drawString(Ssala4+"ºC",((3*c4x-c2x)/2)-17,((c1y+c3y)/2)+7);
            g.drawString("Sala-5",c4x-16,((H-c1y+c3y+10)/2)-5);
            g.drawString(Ssala5+"ºC",c4x-20,((H-c1y+c3y+10)/2)+7);
            g.drawString("Temperatura Ambiente 1",(W/2)-50,22);
            g.drawString(StempAmb1+"ºC",(W/2)-10,34);
            g.drawString("Temperatura Ambiente 2",(W/2)-50,H-14);
            g.drawString(StempAmb2+"ºC",(W/2)-10,H-2);
        }
    }
public void desenha(String s)
```

```
{
   if(!s.equals("NOVO GRAFICO"))
   {
      StringTokenizer tokens = new StringTokenizer(s);
      String Snum = tokens.nextToken();
      String temp = tokens.nextToken();
      temp = tokens.nextToken();
      Bnum = Integer.parseInt(Snum);
      if (Bnum==Pic_1)
      {
         Ssala1 = tokens.nextToken();
         Ssala2 = tokens.nextToken();
         Ssala6 = tokens.nextToken();
         StempAmb1 = tokens.nextToken();
         Isala1 = Float.parseFloat(Ssala1);
         Isala2 = Float.parseFloat(Ssala2);
         Isala6 = Float.parseFloat(Ssala6);
         ItempAmb1 = Float.parseFloat(StempAmb1);
      }
      else
      {
         Ssala3 = tokens.nextToken();
         Ssala4 = tokens.nextToken();
         Ssala5 = tokens.nextToken();
         StempAmb2 = tokens.nextToken();
         Isala3 = Float.parseFloat(Ssala3);
         Isala4 = Float.parseFloat(Ssala4);
         Isala5 = Float.parseFloat(Ssala5);
         ItempAmb2 = Float.parseFloat(StempAmb2);
      }
      this.repaint();
   }
}
public Dimension getPreferredSize()
{
   return new Dimension(400,180);
}
}
```

```
    ************************************Arquivo Útil.Java************************************
import javax.swing.*;
import java.text.DecimalFormat;

public class Util extends Object
{
  public static boolean campoVazio(String s,String S2)
  {
    if(s.equals(""))
    {
      JOptionPane.showMessageDialog(null, "Digite o Valor de "+ S2 ,
          "Campo Vazio !", JOptionPane.ERROR_MESSAGE);
      return(true);
    }
    else
      return(false);
  }

  public static boolean campoCorreto(String s,String S2)
  {
    float emprestimo;
    s = s.replace(',','.');

    try
    {
      emprestimo = Float.parseFloat(s);
      return(true);
    }
    catch(NumberFormatException nfe)
    {
      JOptionPane.showMessageDialog(null, S2+ " Digitado incorretamente !",
      "Dado com Erro !",
          JOptionPane.ERROR_MESSAGE);
      return(false);
    }
  }
   public static short FloatShort(float f)
   {
      f = f*100;
      if (f>32767)
      {
         return(32767);
      }
      else
      {
         short i = (short) Math.floor(f);
         return(i);
      }
   }
   public static short trataTemp(float f)
   {
      f = f*2;
      if (f>255)
      {
         return(255);
      }
      else
      {
         short i = (short) Math.floor(f);
         return(i);
```

```
        }
    }
//O método a seguir converte a frequencia dada pelo usuário na
//metade do período desta frequencia. Além disso, o valor é multiplicado
// por (1/0,035) de modo que cada bit do resultado corresponda a 35 ms.
//Este procedimento foi criado para facilitar a programação da onde de referencia
//no PIC.

    public static short FreToPer(float f)
    {
        double d;
        if (f == 0)
        {
            return(32767);
        }
        else
        {
            d = 1/(f*2*35*0.001);
        }
        if (f>32767) // Não pode ultrapassar este valor já que Java trabalha com tipo "signed"
                // e então com Short(16 bits) o maior valor é 32767.
        {           //Desta maneira o menor valor de frequencia que o usuário pode entrar é com
                    //0.0004.36;
            return(32767);
        }
        else
        {                   //O maior valor de frequencia que o usuário pode entrar é 28.5 Hz
                            //Valores maiores o período será arredondado para zero.
            short i = (short) Math.floor(d);
            return(i);
        }
    }

}
        **********************************TituloPanel.java**********************************
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TituloPanel extends JPanel
{
    private Image logo1;

public TituloPanel(Image image)
{
    logo1 = image;
}
public void paintComponent(Graphics g)
{
    super.paintComponent(g);
    g.drawImage(logo1,0,0,getHeight()+30,getHeight(),this);
    g.setFont(new Font("Serif", Font.BOLD + Font.ITALIC, 40));
    g.setColor(Color.red);
    g.drawString("Experiment - Temperature Control",120,35);
}
public Dimension getPreferredSize()
{
    return new Dimension(800,40);
}
}
```