Flavio André de Carvalho Wolle
Guilherme N. L. Veiga da Rocha

9,0 (nove)

Sistema Supervisório Descentralizado Baseado Em Íntegrons

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo como trabalho de formatura.

São Paulo
1998

Flavio André de Carvalho Wolle
Guilherme N. L. Veiga da Rocha

Sistema Supervisório Descentralizado Baseado Em Íntegrons

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo como trabalho de formatura.

Área de Concentração:
Engenharia de Automação

Orientador:
Prof. Dr. José Reinaldo Silva

São Paulo
1998

Aos nossos pais, sem os quais não chegaríamos a essa etapa da vida, e cujo apoio e dedicação foi fundamental ao longo de todo esse longo percurso

AGRADECIMENTOS

Ao professor e orientador Prof. Dr. José Reinaldo pela condução do trabalho, orientação e ajuda em todos os momentos.

Ao mestrando Nilson Francescotti, pelo imenso apoio e paciência no tocante aos sistemas elétricos do trabalho.

Ao mestrando Pedro Gonzalez del Foyo, pela ajuda, comentários e sugestões apresentados.

Aos amigos Paulo Cesar Debenest e Eduardo Aoun Tannuri pela ajuda na elucidação de vários problemas.

A todos os amigos que deram seu apoio e colaboraram, direta ou indiretamente com o progresso deste trabalho.

Sumário

LISTA DE FIGURAS .....

1. INTRODUÇÃO 9

2. OBJETIVOS 12

3. DESCRIÇÃO DE ALGUNS SISTEMAS DE AUTOMAÇÃO PREDIAL 13

3.1. SISTEMAS DE ILUMINAÇÃO 13
3.2. SEGURANÇA 16
3.3. SISTEMAS DE HVAC (HEATING, VENTILATION AND AIR-CONDITIONING) 19
3.3.1. "Funcionamento do Sistema": *20*
3.3.2. "Chillers": *21*
3.3.3. "Sistemas de Volume de Ar Variável": *22*
3.3.4. "Acionamento do motor dos ventiladores": *23*
3.3.5. "Sistema de controle": *24*

4. DESCRIÇÃO FUNCIONAL DO SISTEMA DE CONTROLE 26

4.1. SISTEMA DE SEGURANÇA 26
4.2. SISTEMA DE ILUMINAÇÃO 28
4.3. SISTEMA DE VENTILAÇÃO 28

5. MODELAGEM DO SISTEMA 30

5.1. SISTEMA DE SEGURANÇA 31
5.1.1. Controle de acesso *31*
5.1.2. Detecção de incêndio *32*
5.1.3. Alarme *34*
5.2. SISTEMA DE ILUMINAÇÃO 35
5.3. SISTEMA DE VENTILAÇÃO 36
5.3.1. Atuação na velocidade do ventilador *37*
5.3.2. Informação da velocidade do ventilador *38*

6. INTEGRAÇÃO ENTRE OS SUBSISTEMAS 40

7. IMPLEMENTAÇÃO 48

8. DESENVOLVIMENTO DO SOFTWARE DE CONTROLE 50

8.1. COMPONENTES DO SOFTWARE DE CONTROLE: 50
8.1.1. DDE Server: *52*
8.1.2. Módulo Supervisório: *52*
8.1.3. Módulo de controle (CLP): *53*
8.2. ESTRUTURA DOS MÓDULOS SUPERVISÓRIO E DE CONTROLE: 54
8.3. COMUNICAÇÃO ENTRE OS MÓDULOS: 57
8.3.1. Verificação da comunicação pelo supervisor: *57*
8.3.2. Verificação da comunicação pelo módulo de controle (CLP): *58*

ANEXO A - DESCRIÇÃO DO MÓDULO SUPERVISÓRIO....61

A.1. TELA SETUP PARAMETERS:....62
A.2. TELA CONTROL PANEL:....63
A.3. TELA DE HISTÓRICO DE TEMPERATURA E VELOCIDADE:....65

BIBLIOGRAFIA....66

APÊNDICE I – LIGAÇÕES ELÉTRICAS....67

I.1. FONTES DE ALIMENTAÇÃO....67
I.2. COMPONENTES....68
I.3. CONSIDERAÇÕES SOBRE POTÊNCIA E FILTRAGEM....69

APÊNDICE II - PWM....71

APÊNDICE III - DESCRIÇÃO DAS FUNÇÕES DO MÓDULO DE CONTROLE (CLP)....73

III.1. FUNÇÃO FCNCOMUN:....74
III .2. FUNÇÃO FCNLDTEMP:....75
III .3. FUNÇÃO FCNINTRU:....75
III .4. FUNÇÃO FCNALARM;....76
III .5. FUNÇÃO FCNVENTIL....79
III .6. FUNÇÃO FCNARRAY:....80
III .7. FUNÇÃO SPDCONVERT E TEMPCONVERT:....83

APÊNDICE IV - FLUXOGRAMA DO CICLO DE CONTROLE DO MÓDULO SUPERVISOR:..84

APÊNDICE IV - LISTAGEM DAS FUNÇÕES DO MÓDULO DE CONTROLE (CLP):....86

IV.1. PROGRAMA PRINCIPAL....86
IV.2. FUNÇÃO FCNALARM;....89
IV .3. FUNÇÃO FCNARRAY;....90
IV .4. FUNÇÃO FCNCOMUN;....96
IV .5. FUNÇÃO FCNINTRU;....96
IV .6. FUNÇÃO FCNLDTEMP;....97
IV .7. FUNÇÃO FCNVENTIL:....97
IV .8. FUNÇÃO SPDCONV;....99
IV .9. FUNÇÃO TEMPCONV;....99

VI. LISTAGEM DO PROGRAMA SUPERVISÓRIO....101

VI.1. SUPERVISOR.UIR....101
VI.2. SUPERVISOR.H....101
VI .3. DEFINES.H....103
VI .4. FUNCTIONS.H....104
VI .5. GBLVARS.H....104
VI .6. SUPERVISOR.C....105

Lista de Figuras

1.1	Hierarquia da arquitetura de sistemas baseados em íntegrons	10
5.1	Rede de controle de acesso	32
5.2	Rede de detecção de incêndio	33
5.3	Rede de acionamento do alarme	34
5.4	Rede de acionamento da iluminação	36
5.5	Rede de ajuste da velocidade do ventilador	38
5.6	Rede de atualização da velocidade do ventilador	39
6.1	Sistema de Intrusão	40
6.2	Sistema de Iluminação	41
6.3	Sistema de Iluminação	41
6.4	Sistemas de detecção de incêndio e alarme de incêndio	42
6.5	Sistema de Ventilação	44
6.6	Sala desocupada	44
6.7	Rede representativa do sistema supervisório	45
6.8	Modelo do programa de controle do CLP	46
6.9	Sistema CLP-Supervisório	47
7.1	Esquema da Maquete a ser Implementada	49
8.1	Relação entre os módulos de software do sistema de controle	51
A.1	Tela do *Setup Parameters*	62
A.2	Tela do Control Panel	63
A.3	Tela do *Show History*	65
I.1	Esquema elétrico de uma das salas da maquete	70
II.1	Esquema do PWM	72
III.1	Fluxograma do programa no CLP	73
III.2	Fluxograma da função FCNCOMUN	74
III.3	Fluxograma da função FCNLDTEMP	75
III.4	Fluxograma da função FCNINTRU	76

Lista de Figuras

III.5 Fluxograma da função FCNALARM 78

III.6 Fluxograma da função FCNVENTIL 80

III.7 Fluxograma da função FCNARRAY 82

IV.1 Fluxograma do cliclo do supervirsos 85

1. Introdução

Atualmente, a integração de sistemas é vista como uma estratégia capaz de garantir grande economia decorrente de uma maior eficiência na utilização dos recursos de um sistema. Esse aumento de eficiência é possível graças a um controlador capaz de coordenar o uso de cada um dos recursos disponíveis de forma que ocorra uma cooperação entre os sub-sistemas, resultando assim num aumento de produtividade e qualidade do sistema como um todo.

Entretanto, em grande parte dos sistemas atualmente em funcionamento, o que se observa não são sistemas integrados propriamente ditos, mas sim uma série de sub-sistemas trabalhando isoladamente, sendo cada um deles controlado de maneira independente.

Em alguns casos, existem sistemas de coleta de dados que os fornecem a operadores humanos, encarregados de fazer os ajustes necessários de forma a garantir um funcionamento adequado do sistema. Contudo, esses operadores nem sempre são especialistas, o que acaba por reduzir o desempenho do sistema, tornando o emprego de um sistema automático de controle uma alternativa atraente.

Para a implementação de um sistema realmente integrado, é necessário que se realize uma formalização do sistema que leve em conta tanto sua arquitetura, como os processos de tomada de decisões envolvidos. Uma vez tendo-se um modelo formal do sistema, é possível se criar ferramentas que permitam a simulação do comportamento do sistema, o que auxiliará o projeto do sistema de controle. Além disso, o modelo deve permitir a monitoração do sistema sendo controlado.

No modelo empregado nesse trabalho, a arquitetura do sistema envolverá dois níveis hierárquicos: a planta e o supervisório. O nível da planta engloba os subsitemas e seus controladores especializados, que executam as funções de controle de forma assíncrona. O supervisório corresponde à parte "inteligente" do controle, sendo responsável pela coordenação das ações dos controladores no nível da planta.

Cada conjunto formado por uma planta e um sistema supervisório correspondente forma uma unidade denominada *íntegron*. Diversos *íntegrons* podem estar subordinados a um sistema supervisório de hierarquia mais alta, formando um *íntegron* de ordem superior. Como pode-se notar, esta é uma estrutura recursiva e hierárquica, cuja implementação pode-se dar através de sistemas cliente-servidor.

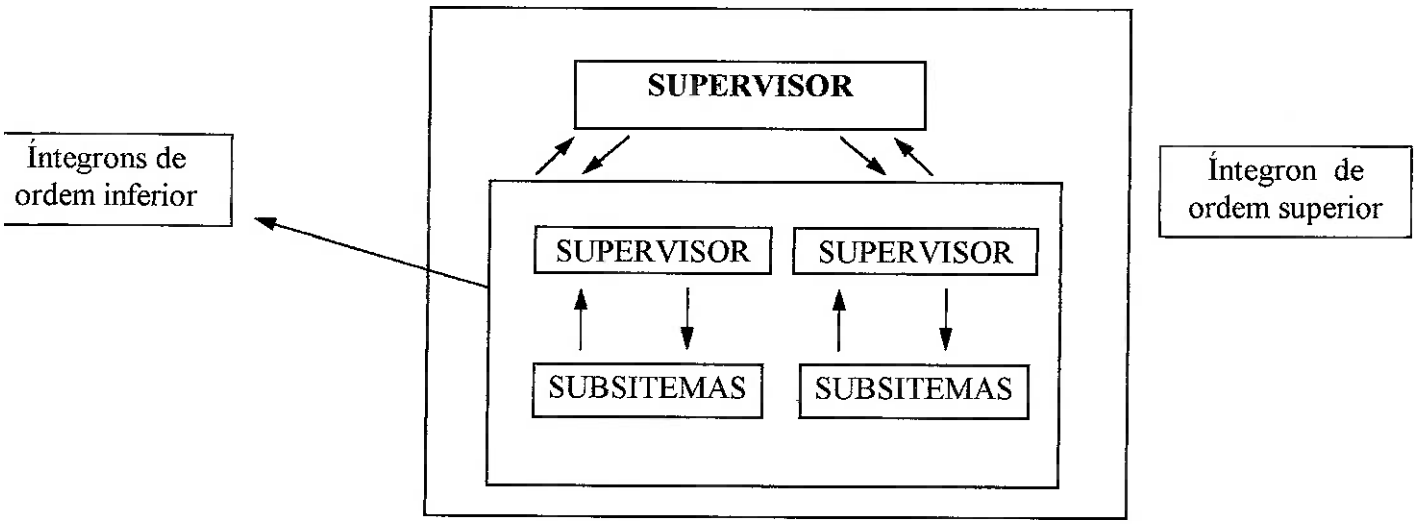


Figura 1.1 - Hierarquia da arquitetura de sistemas baseados em íntegrons

Um importante aspecto da divisão do sistema nesses dois níveis hierárquicos diz respeito à quantidade de dados que circulam no sistema. Como o supervisório se preocupa apenas com a coordenação das ações dos subsitemas, não é necessário se transmitir todos os dados de cada subsistema, mas apenas aqueles necessários para a coordenação das atividades. Isso também torna a operação do sistema mais simples, uma vez que uma interface de alto nível entre o sistema de controle o os operadores pode ser criada.

O nível da planta é geralmente constituído tanto por sistemas de variáveis contínuas como por sistemas a eventos discretos, enquanto o supervisório pode ser representado apenas por sistemas a eventos discretos. Quando os dois tipos de sistemas estão presentes, temos um sistema de controle híbrido. Nesse caso, uma maneira de se simplificar a modelagem do sistema consiste em buscar uma forma de "discretizar" as parcelas contínuas do sistema.

Uma forma de se representar sistemas a eventos discretos é através de redes baseadas em Redes de Petri, compostas por elementos de natureza estática (estados) e dinâmica (eventos). Nesse trabalho, será utilizada uma forma de Rede de Petri estendida: a rede GHENeSys (General Hierarchical Enhanced Net System) proposta por Silva e Miyagi [7]. Esse tipo de rede é mais adequada para a representação de um sistema hierárquico. Isso se deve ao fato de que, nesse tipo de rede, uma série de estados e eventos sejam representados por um único elemento.

2. Objetivos

O presente trabalho objetivará o desenvolvimento de um sistema que forma um *integron*, que será parte de um sistema maior e mais complexo, sua implementação e realização de testes do sistema de controle projetado;

O sistema a ser desenvolvido será composto de uma maquete que procurará simular os ambientes de duas salas. Cada uma das salas terá seus próprios sensores e será controlada por um CLP próprio, exatamente por que o comportamento das salas é independente um do outro.

3. Descrição de alguns sistemas de Automação Predial

Para a especificação da implementação do sistema de controle para o ambiente a ser simulado, foi feita uma pesquisa de como os sistemas envolvidos são comumente tratados. Cada um dos sistemas envolvidos será visto a seguir.

3.1. Sistemas de Iluminação

O controle de iluminação de forma automática se mostra vantajoso em vários sentidos, como permitir um maior conforto ao usuário, gerir de forma mais eficiente o uso de energia, reduzindo os recursos gastos com a energia elétrica.

Na medida em que o controle inteligente das luzes permite que espaços desocupados ou fora de uso não tenham suas luzes acesas desnecessariamente, temos assim já uma melhora no combate ao desperdício. É pertinente dizer que o momento em que a automação predial é discutida é uma excelente oportunidade para se discutir e adotar um sistema mais moderno e eficiente de iluminação.

Quando o ambiente predial tem seu ritmo de uso alterado (por exemplo com expansões ou reduções, ou quando se fazem reformas no prédio), o sistema de iluminação pode se tornar inadequado. Este é um outro bom momento para se aplicar o controle inteligente e aplicar melhorias no sistema de iluminação.

Muitos métodos podem ser utilizadas na melhoria da performance dos sistemas de iluminação: o uso de softwares EMC (Energy Management and Control Systems), sensores de iluminação que permitem verificar se o nível de iluminação natural é suficiente ou não, e caso não seja, determinar qual o nível de iluminação artificial é necessário, e finalmente sensores de presença, que evitam que se acendam as luzes em locais não

utilizados. A instalação dos sensores de presença, além de servirem ao sistema de gerenciamento de energia, se mostram úteis na implementação de um sistema de segurança.

Normalmente os sensores de presença utilizam detecção por infravermelho, emissões de ultrassom, microondas ou por som. Os dois primeiros são os mais populares e são largamente utilizados, e sensores que combinam ambos estão disponíveis, aliando o melhor de cada um deles. Normalmente esses sensores são instalados diretamente em uma parede ou no teto do ambiente a ser verificado.

Os sensores de infravermelho detectam o movimento de fontes de calor (como o movimento de uma pessoa) em frente ao sensor. Pequenos movimentos não são detectados por esse tipo de sensor (como por exemplo o movimento de uma pessoa digitando em um computador). Esses sensores são úteis quando se tem um área livre, e se quer detectar por exemplo a entrada de pessoas em uma sala. Porém não são utilizavéis para detectar a presença de objetos parados.

Os sensores de ultrassom emitem uma onda de freqüência entre 25 e 40Khz, bem acima portanto daquela percebida pelo ouvido humano. Objetos se movendo, mesmo que não diretamente na frente dos sensores causam mudança no sinal que retorna ao sensor. Esses sensores são bastante sensíveis, detectando pequenos movimentos e podem ser disparados até pelo movimento de cortinas em uma sala. São úteis para a detecção em ambientes obstruídos (onde não se pode ter uma movimentação livre).

Um dos principais parâmetros de escolha do sensor é a área de cobertura do sensor. Os fabricantes fornecem especificações detalhadas nesse sentido. O comportamento dos sensores instalados varia bastante com o ambiente, portanto é

recomendado que se façam diversos testes com diferentes configurações para se determinar a disposição ideal.

Para uso em automação predial pode-se usar um sensor de presença do tipo acionamento-manual/desligamento-automático, pois assim uma pessoa em uma sala pode desligar ou ligar as luzes conforme queira, mas garante que quando não houverem pessoas na sala as luzes permanecerão desligadas.

Há também os sensores de depreciação do "lúmen" da lâmpada, que procuram manter sempre o mesmo nível de iluminação dado pela lâmpada, pois as lâmpadas normalmente emitem menos luz conforme o tempo de uso aumenta. Isso faz com que no início do uso de uma lâmpada, a sua iluminação seja maior do que a especificada em projeto.

Estudos do Electric Power Research Institute dos EUA mostram que o controle de presença permite alcançar economias de energia da ordem de:

- 25% em escritórios particulares;
- 35% em salas de conferência;
- 40% em dormitórios;
- 65% em salões de conferência em hotéis.

O projeto de sensores também deve considerar a calibração destes para que não haja desligamento e ligamento freqüentes, o que causa a diminuição da vida útil das lâmpadas.

Portanto, verificamos a importância do controle do sistema de iluminação, já que pode trazer economias da ordem de 30~50% no gasto de energia, que representam boa

parte dos gastos com energia elétrica de um ambiente predial, o que tem óbvio impacto financeiro e ambiental.

Porém o conforto do usuário não deve ser deixado de lado, pois do contrário irá prejudicar o desempenho de um funcionário, diminuir a satisfação do usuário, que podem levá-lo até mesmo a sabotar os controles, gerando perda material e de eficiência no sistema.

3.2. Segurança

O sistema de segurança é uma das principais aplicações da automação predial, onde se pode integrar os diversos sistemas necessários à sua implantação com os outros sistemas, como alarmes e iluminação.

Basicamente tratar-se-á do problema de invasão, ou seja, da entrada não autorizada de uma pessoa em um ambiente monitorado. Essa invasão pode ter como causa a entrada de pessoa em horário não autorizado, em local onde não pode ter acesso ou em local desativado para uso.

Um sistema simples de detecção de invasão pode utilizar-se da estrutura de sensores já vista para a implementação do sistema de iluminação, onde sensores de presença estão dispostos dentro do ambiente a ser monitorado de forma a indicar se o ambiente está ou não ocupado, ou se acontece a entrada ou saída de uma pessoa. Os sensores de presença já foram discutidos anteriormente, e portanto este tópico não será revisitado.

O evento de invasão pode ter como tratamento simples o acionamento de um alarme, como é comum em residências ou pequenos estabelecimentos comerciais. São

bastante comuns os sistemas fechados que realizam somente essa função e que podem ser comprados facilmente. Mas nesse caso o sistema de segurança permanece isolado dos demais sistemas, o que não nos interessa.

Pode-se acrescentar mais sensores aos já mencionados, como sensores que detectam a abertura de portas ou janelas, que não acrescentam grande complexidade ao sistema de segurança.

Neste sistema simples, a única ação planejada em caso de invasão é o acionamento de alarme sonoro, que pode avisar, no caso de uma residência, os vizinhos ou um vigilante ou até mesmo uma patrulha policial. A fim de sofisticar mais essa ação, pode-se fazer com que a detecção da invasão acione um serviço privado de vigilância, ou até mesmo uma central de polícia (como é comum em bancos e lojas visadas a assaltos, como joalherias).

Pode-se acrescentar ao sistema um circuito fechado de televisão, que pode realizar o constante monitoramento do ambiente, e que pode armazenar imagens de uma sala quando está tiver um acesso não autorizado. No caso de haver uma equipe de vigilância responsável pelo acompanhamento do sistema de segurança, pode-se deixar a cargo desta a ação apropriada em caso de invasão.

Sistemas mais sofisticados incluiriam a utilização de um sistema de banco de dados, onde seriam armazenadas as informações relativas aos funcionários e usuários do estabelecimento, onde poderiam se guardar informações a respeito dos horários e dos locais aos quais uma pessoa tenha acesso. A verificação do acesso então pode ser feita com utilização de cartões magnéticos ou ainda, por cartões inteligentes (*smart cards*) e dispositivos de leitura desses cartões. Um sistema de senhas pode ser utilizado para impedir (ou ao menos tentar impedir) o uso não autorizado desses cartões. Poderia ainda

se pensar em outros meios de validação desses cartões, como por exemplo utilizando o banco de dados para armazenar imagens dos funcionários, e verificar se o usuário do cartão é realmente a pessoa a quem ele foi designado.

Ao sistema de detecção de invasão pode ser acrescentado o sistema de detecção de incêndio, que também zela pela segurança dos usuários de um ambiente predial.

Novamente diversas soluções podem ser estudadas. O passo inicial para a implantação deste sistema é a forma de como um incêndio pode ser detectado. Normalmente, sensores de temperatura podem ser utilizados para essa função, verificando se uma determinada temperatura considerada alta o suficiente para indicar se uma situação de alerta foi atingida. No entanto, por si só uma temperatura alta não indica que um incêndio esteja ocorrendo (nada impede que uma pessoa, mal intencionada ou até mesmo sem intenção, acenda um isqueiro próximo a um sensor de temperatura, produzindo um alerta desnecessário). Pode-se então verificar se essa alta temperatura é detectada em mais de um ponto (pode-se instalar mais de um sensor de temperatura em uma sala) ou verificar se essa alta temperatura afeta outras salas vizinhas.

Uma outra maneira seria associar uma alta temperatura com o sinal proveniente de um sensor de fumaça. Nesse caso, um sinal de alta temperatura combinado com a presença de fumaça poderia disparar um sinal que indica incêndio. Nada impede que as diversas soluções apresentadas sejam utilizadas em conjunto.

Com relação às ações a uma ocorrência de incêndio, também podemos encontrar várias alternativas: pode-se acionar um sistema de *sprinklers*, que nada mais são do que jatos de água instalados no teto de uma sala. Poderia se integrar o sistema de detecção de incêndio com o de segurança, e assim uma equipe de vigilância poderia através do circuito

interno de televisão verificar a situação de uma sala de onde partiu o sinal de incêndio, podendo acionar o Corpo de Bombeiros ou até mesmo uma brigada interna de combate a incêndio.

Como pode-se observar, a complexidade do sistema de segurança pode variar enormemente, conforme os recursos (financeiros, tecnológicos e humanos) disponíveis, tanto para a implantação quanto para a manutenção do sistema.

A discussão de como os sistemas de segurança podem ser implantados poderia gerar por si só um extenso trabalho, e fugiria ao escopo deste trabalho um estudo mais detalhado sobre esse assunto.

3.3. Sistemas de HVAC (Heating, Ventilation and Air-Conditioning)

Segundo dados do governo americano, os sistemas de HVAC (Heating, ventilation and air-conditioning) estão entre os maiores consumidores de energia em prédios públicos americanos. Esses sistemas são responsáveis principalmente por manter o conforto térmico dentro de edifícios. Para isso, envolvem sistemas de aquecimento, refrigeração, controle de umidade, filtragem de ar e renovação de ar. Os equipamentos envolvidos para a execução dessas funções são torres de refrigeração ("chillers"), sistemas de volume de ar variável, aquecedores ("boilers") e armazenadores de vapor ("steam traps").

Como o consumo de energia nesses sistemas é bastante elevado, justifica-se um maior investimento em medidas que acarretem uma maior economia durante a fase de operação do sistema. Uma das formas de se atingir uma maior economia durante a operação desse tipo de sistema é através de um sistema de controle eficiente. A seguir, serão descritos os componentes envolvidos na parte de refrigeração do sistema de HVAC

e, em seguida, as medidas de controle que podem ser tomadas de forma a garantir maior eficiência durante a sua operação.

3.3.1. "Funcionamento do Sistema":

Antes que se faça a descrição de cada componente separadamente, é necessário se descrever o funcionamento do sistema como um todo, de forma a se saber qual o papel de cada um dos componentes no funcionamento global do sistema.

No sistema de refrigeração, as torres de refrigeração ("chillers") são responsáveis por resfriar uma massa de líquido refrigerante (como por exemplo a água). Uma parte dessa massa de líquido frio é então enviada a trocadores de calor que resfriam o ar que será enviado ao ambiente a ser refrigerado.

Esses trocadores de calor podem ajustar a quantidade de líquido refrigerante utilizada conforme a temperatura do ar esteja mais alta ou mais baixa. Entretanto, a vazão dentro do "chiller" é mantida constante, pois sua alteração resultaria em perda de eficiência.

Próximo aos trocadores de calor estão instalados ventiladores de grande porte responsáveis por estabelecer um fluxo de ar que passe sobre os trocadores de calor e prossiga então para os ambientes a serem refrigerados. A velocidade dos ventiladores é regulada de forma a garantir que o fluxo de ar esteja de acordo com a demanda.

Antes de passar para o interior do recinto refrigerado, o fluxo de ar encontra uma restrição - os *dampers* - que permitem que se ajuste a quantidade de ar frio demandada. Caso o ambiente precise de maior refrigeração, os *dampers* são abertos, caso contrário são fechados.

3.3.2. "Chillers":

Um dos aspectos que deve ser levado em conta ao se fazer ajustes que permitam economia de energia nos "chillers" é o impacto que essa alteração pode acarretar no sistema de refrigeração como um todo.

Uma importante característica desse componente é o fato de a temperatura do líquido refrigerante enviado aos trocadores de calor não poder ser alterada. Isso ocorre porque, caso a temperatura desse líquido seja elevada para permitir um menor consumo de energia, ocorrem alterações na umidade dos ambientes que acabam por reduzir a eficiência do sistema de refrigeração.

Um outro limitante nos ajustes que podes ser feitos nesse componente é a necessidade de manter o fluxo de líquido refrigerante constante. Os *chillers* são projetados para operar num determinado fluxo de líquido refrigerante e o seu funcionamento fora do fluxo de projeto gera ineficiências. Dessa forma, mesmo que a demanda por líquido refrigerante diminua, não se deve reduzir o fluxo no "chiller" de forma a não comprometer o bom funcionamento do sistema.

Sem desconsiderar as restrições descritas acima, existem medidas que permitem garantir uma economia de energia na utilização dos *chillers* dentro do sistema de refrigeração. A seguir, são descritas algumas dessas medidas.

A primeira dessas medidas consiste no emprego de vários *chillers* de diferentes capacidades. Durante os períodos de baixa demanda, os *chillers* de menos capacidade são mantidos em funcionamentos. Conforme a demanda aumenta, devido, por exemplo, ao

aumento da temperatura externa ou ao aumento do número de pessoas no interior do edifício, os *chillers* de maior capacidade são postos em funcionamento. Dessa forma, não se gasta energia fazendo o resfriamento de uma massa de fluído que não será enviada aos trocadores de calor responsáveis pela refrigeração do ar.

Uma outra forma de se utilizar os *chillers* de forma mais econômica consiste em fazê-los funcionar durante horas de menor demanda para gerar uma massa de gelo que, durante as horas de maior demanda, é utilizada para auxiliar o sistema de refrigeração. A economia desse método decorre do fato de a energia nas horas de menor demanda ser mais barata. Além disso, isso permite que os *chillers* funcionem de forma mais regular durante todo o período e, com isso, podem ser dimensionados não para os momentos de pico, mas para a demanda média do período.

3.3.3. "Sistemas de Volume de Ar Variável":

O sistema descrito acima se trata de um sistema de volume de ar variável, ou seja, um sistema onde a velocidade do ventilador é ajustada de forma a garantir um fluxo de ar adequado à demanda no momento. Esse sistema permite que os *dampers* funcionem em posições próximas à posição totalmente aberta, o que reduz a dissipação de energia nos *dampers*.

Existem sistemas onde o fluxo de ar sobre os trocadores de calor é mantido constante (sistemas de volume constante) independentemente da demanda de refrigeração. O ajuste é feito apenas através dos *dampers* o que resulta em perda de energia, pois em situações de bixa demanda, os dampers serão mantidos em posição próxima a totalmente

QUE MERDA !!!

fechada. Com isso, a energia utilizada para manter o fluxo constante é em grande parte dissipada nos *dampers*.

3.3.4. "Acionamento do motor dos ventiladores":

Um outro meio de se economizar energia nos sistemas de refrigeração é se fazer o acionamento dos motores envolvidos de maneira mais eficiente. A seguir se faz uma breve descrição de alguns desses métodos:

- Modulação em largura de pulso (PWM): é o método mais utilizado para motores entre ½ HP e 500HP devido a seu baixo preço, confiabilidade e disponibilidade.
- Inversores de fonte de corrente (CSI): são bastante confiáveis graças a simplicidade de seu circuito e suas caracterísiticas de limitação de corrente. Além disso, permitem a implementação de regeneração de energia.
- Inversores de fonte de tensão(VSI): são semelhantes aos inversores CSI, mas utilizam a tensão em lugar da corrente.

3.3.5. "Sistema de controle":

O sistema de controle têm um papel fundamental na economia de energia no sistema de ventilação. A partir das variáveis de estado de cada um dos componentes envolvidos no sistema de refrigeração, o sistema de controle deve ser capaz de ajustar os parâmetros dos atuadores envolvidos de forma a garantir o melhor funcionamento do sistema.

No sistema descrito acima, o sistema de controle é o responsável por integrar os diversos componentes. Por exemplo, ele deve ser capaz de determinar em que

momento a velocidade dos ventiladores instalados junto aos trocadores de calor deve ser ajustada. Para isso, deve verificar os estados dos *dampers* de cada sala ligada àquele duto. Caso grande parte deles esteja próxima à posição totalmente fechada, a velocidade do ventilador deve ser reduzida de forma a não se desperdiçar energia. Além disso, em sistemas onde *chillers* de diferentes capacidades estejam instalados, é o sistema de controle o responsável por, a partir do fluxo demandado pelos trocadores de calor, determinar quais deles devem estar em operação.

Além de realizar esse tipo de função, o sistema de controle deve levar em conta outros aspectos, como por exemplo, a quantidade de pessoas presentes no edifício e a temperatura externa, de forma a melhorar o desempenho do sistema de refrigeração. Para isso, o sistema de controle pode apresentar diferentes graus de complexidade.

Uma forma simples de controle consiste em manter o sistema em diferentes níveis de funcionamento conforme haja ou não presença de pessoas num ambiente. No momento em que não houver ninguem em um recinto, o sistema pode funcionar em um nível fixo, reduzido de atividade. Uma vez que alguém entre nessa sala, o sistema passa a controlar a temperatura de forma a satisfazer a temperatura de conforto ajustada pelo usuário.

Um controle mais refinado pode, por exemplo, em lugar de manter um nível fixo no momento em que não houver ninguem na sala, verificar as condições atuais e a partir daí ajustar o nível mais adequado de funcionamento do sistema. O sistema pode levar em conta a temperatura atual da sala, a temperatura externa, a presença de janela na sala ou não, entre outras variáveis. O nível de complexidade do sistema será tanto maior quanto maior for o número de variáveis levadas em conta.

TCC - SISTEMA SUPERVISÓRIO DESCENTRALIZADO BASEADO EM INTEGRONS.

PG - 9, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25.

Uma outra decisão que caberia a um controle mais sofisticado seria decidir em que momentos é economicamente vantajoso se produzir uma massa de gelo utilizando a capacidade ociosa do sistema para posterior uso. Essa decisão envolveria aspectos como o preço da energia no momento, o nível de ociosidade do sistema, uma previsão de em quanto tempo essas reservas passariam a ser usadas, entre outros.

O aumento do custo decorrente da utilização de sistemas mais complexos de controle resulta principalmente do aumento do número de sensores necessários para se ter dados mais precisos sobre o estado do sistema. A implementação de algoritmos mais complexos pode ser feita de forma relativamente fácil e barata por envolver apenas a alteração de software.

4. Descrição Funcional do Sistema de Controle

O primeiro passo para se desenvolver um sistema de controle (não somente de um sistema predial, mas de qualquer sistema) é a descrição de como o sistema deve se comportar. Deve-se pensar em quais parâmetros estarão envolvidos e qual deve ser o comportamento do sistema em função das variações desses parâmetros.

A seguir, são apresentados os parâmetros envolvidos em cada um dos subsistemas a ser controlado e o comportamento esperado.

4.1. Sistema de Segurança

O sistema de segurança é responsável pela verificação de situações não autorizadas ou que possam afetar a segurança das pessoas e/ou instalações do ambiente em questão.

No ambiente que está sendo projetado haverá duas preocupações quanto à segurança: controle do acesso às salas e detecção de incêndio.

No tocante ao acesso das salas, o sistema permitirá que elas sejam habilitadas ou não, conforme se planeje a sua utilização normal. No sistema que está sendo implementado, a presença de uma pessoa numa sala num momento em que ela esteja desabilitada será encarada como uma intrusão e essa situação será transmitida a um supervisório de ordem superior. A forma como essa intrusão será tratada fica a cargo desse nível superior de controle. Isso permite uma certa flexibilidade quanto às medidas que podem ser tomadas, conforme os recursos disponíveis. O supervisório superior pode fazer verificações adicionais para uma melhor definição do problema, podendo acionar um sistema de alarme, alertar um serviço de vigilância ou até contatar uma central de polícia. O acionamento de alarme na própria sala não foi previsto pelo sistema de forma a permitir

que o problema ocorrido seja esclarecido, definindo se é uma pessoa estranha ao ambiente ou se é um funcionário fora de seu horário normal de trabalho, uma equipe de manutenção. Essa identificação poderia ser feita, por exemplo, através de um circuito interno de televisão.

Será considerada uma situação de incêndio quando for detectada concomitantemente a presença de fumaça na sala (através de um sensor apropriado) e um histórico de temperatura e nível de demanda do sistema de ventilação crescente nos últimos 10 minutos.

Quando em condições normais, o tratamento dessa situação também deve ser feito em níveis superiores, para que as medidas mais adequadas possam ser tomadas, já que nesses níveis pode-se angariar informações mais amplas sobre o problema, levando-se em conta por exemplo a extensão da situação de incêndio ou tipo da área afetada. Conforme a extensão do problema ou a área que foi afetada pode-se decidir simplesmente acionar "sprinklers" de uma sala, avisar uma brigada local de incêndio, ou em caso de maior gravidade acionar o serviço de bombeiros.

Quando as evidências de incêndio ocorrerem em um momento em que a comunicação entre o nível de controle mais baixo e seu supervisor estiver interrompida, o nível inferior deve atuar de forma autônoma, acionando os dispositivos de incêndio sob seu controle. Isso permitirá que medidas de combate ao possível incêndio sejam tomadas ainda que os meios de comunicação entre os diferentes níveis de controle estejam comprometidas.

4.2. Sistema de Iluminação

A iluminação das salas será acionada quando houver presença de uma pessoa num ambiente que esteja habilitado. Conforme visto acima, esse tipo de medida, embora simples, pode acarretar grande economia de energia.

4.3. Sistema de Ventilação

O sistema a ser implementado é uma simplificação de um sistema real de conforto térmico de um ambiente predial, que envolve instalações que vão muito além de uma única sala. Neste sistema em uma das salas haverá um ventilador, que estará fazendo o papel do conjunto "ventilador-*dampers*" (conforme visto em 3.3) de um sistema real. Foi escolhida essa alternativa tanto porque o ventilador permite a atuação sobre a temperatura da sala, como porque a utilização de um ventilador permite que se implemente um sistema de controle do ventilador que se estende ao controle do ventilador do sistema real.

O funcionamento do sistema de ventilação permitirá que o usuário defina uma temperatura (entre 20°C e 26°C). Sempre que a sala estiver habilitada e houver presença na sala, o sistema será acionada de forma a manter a temperatura em uma faixa de 1°C em torno da temperatura definida pelo usuário. Caso não haja ninguém na sala, o sistema funcionará em velocidade constante apenas para evitar um gasto excessivo de energia quando o sistema for acionado.

O sistema de ventilação também deve enviar informações sobre o seu nível de funcionamento para o supervisório de nível superior. Isso permitirá que o sistema de refrigeração como um todo funcione de forma mais eficiente. Por exemplo, no caso de

várias salas estarem funcionando com seus *dampers* em posições de pequena abertura, pode-se atuar no fornecimento de ar frio a fim de diminuir o dispêndio de energia. Isso pode ser feito através da redução da velocidade do ventilador ou por meio de uma diminuição do fluxo do fluido refrigerante nos trocadores de calor do sistema de condicionamento de ar.

5. Modelagem do Sistema

A modelagem do sistema objetiva permitir a descrição do funcionamento do sistema segundo modelos matemáticos que podem ser empregados no desenvolvimento de um software de controle.

Como mencionado acima, a modelagem do sistema será feita de acordo com o modelo GHENeSys, baseado em redes de Petri. A modelagem consistirá então da construção de redes que representem o comportamento do sistema. Cada sistema será modelado separadamente de forma a facilitar o seu estudo. O fato de cada sistema ser modelado independentemente não significa que estejam isolados uns dos outros, pois na própria modelagem aparecerão variáveis comuns a eles, que serão provenientes de recursos compartilhados entre si.

Além disso é importante ressaltar que algumas das variáveis envolvidas são de natureza contínua, sendo portanto necessário a sua discretização de forma a permitir a sua modelagem em forma de redes. A forma como essa discretização é específica para cada sistema e seu respectivo comportamento. Dessa forma a sua discussão será feita junto à sua implementação.

As variáveis provenientes dos sensores de presença e fumaça, assim como as lâmpadas são do tipo discreto. As variáveis vindas dos sensores de temperatura, da velocidade do ventilador e a temperatura definida pelo usuário são do tipo contínuo.

Por fim, levando-se em conta que o sistema a ser modelado deve ser voltado para a integração com sistemas em diferentes níveis hierárquicos, deve-se considerar além das variáveis tradicionais de entrada e saída (relacionadas respectivamente a sensores e

atuadores) as variáveis de troca de informações e comandos entre os diferentes níveis de controle. Para efeito de modelagem as informações provenientes de sensores da planta e de outros níveis hierárquicos podem ser encaradas como entradas enquanto que as informações que serão enviadas a outros níveis e as variáveis de atuação serão encaradas como saídas.

As variáveis serão agrupadas em quatro grupos para fins descritivos:

- Variáveis de entrada: correspondem aos sinais dos sensores;
- Variáveis de saída: correspondem aos sinais enviados aos atuadores;
- Variáveis de informação: são variáveis enviadas a níveis superiores de controle;
- Variáveis de comando: são variáveis recebidas de níveis superiores de controle.

5.1. Sistema de segurança

O sistema de segurança na realidade é composto de uma série de sub-sistemas descritos abaixo:

5.1.1. Controle de acesso

O controle de acesso deve ser capaz de detectar a presença em uma sala e verificar se essa presença é permitida ou não. As variáveis desse sistema são:

variáveis de entrada: presença, coletada pelo sensor de presença.

variáveis de saída: não se aplicam.

variáveis de informação: presença, intrusão.

variáveis de comando: habilitação.

O controle de acesso pode ser representado pela rede mostrada na figura 5.1.

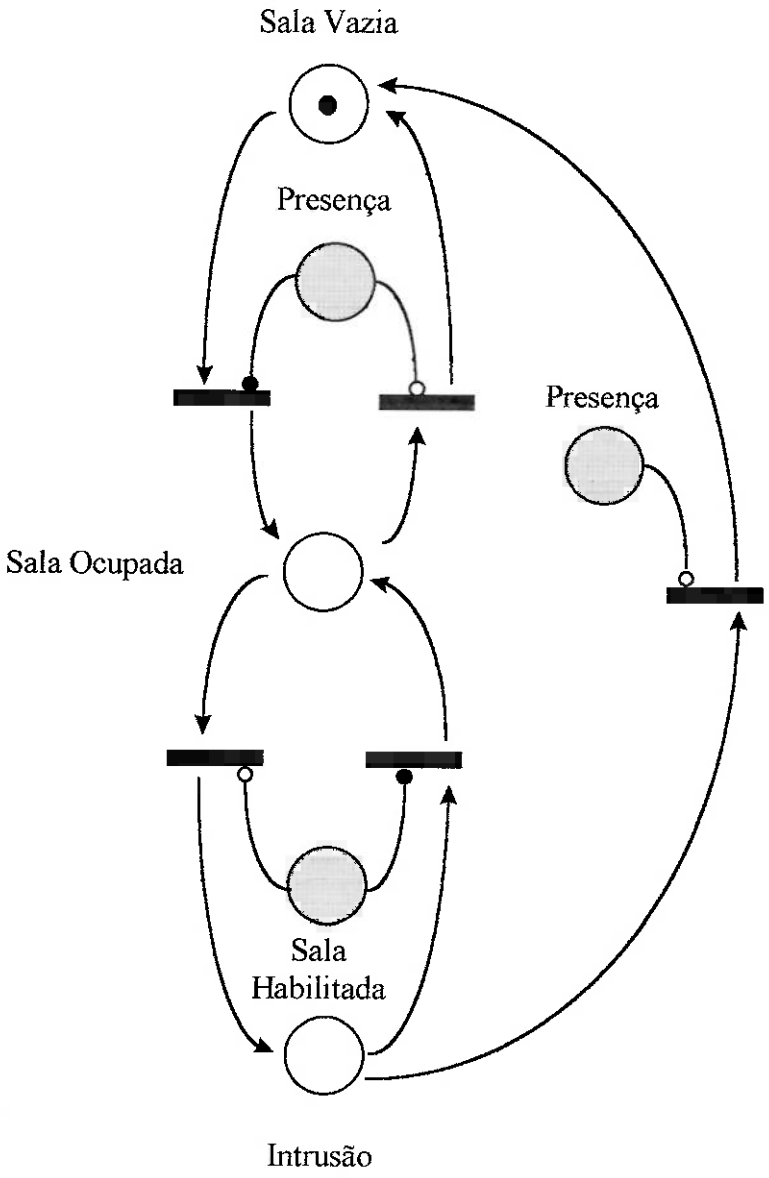


Figura 5.1 - Rede de controle de acesso

5.1.2. Detecção de incêndio

A detecção de incêndio deve informar a um nível superior um estado de incêndio se algumas condições forem encontradas: histórico de temperaturas e velocidade do ventilador crescentes e presença de fumaça. A rede que representa o sistema de detecção de incêndio é mostrada em 5.2.

variáveis de entrada: temperatura, fumaça (provenientes de um sensor de temperatura e de um sensor de fumaça) e estado da comunicação com o supervisor.

variáveis de saída: acionamento dos *sprinklers*, quando a comunicação com o supervisório estiver comprometida.

variáveis de informação: incêndio na sala, quando houver comunicação com o supervisor.

Variáveis de comando: situação normal reestabelecida.

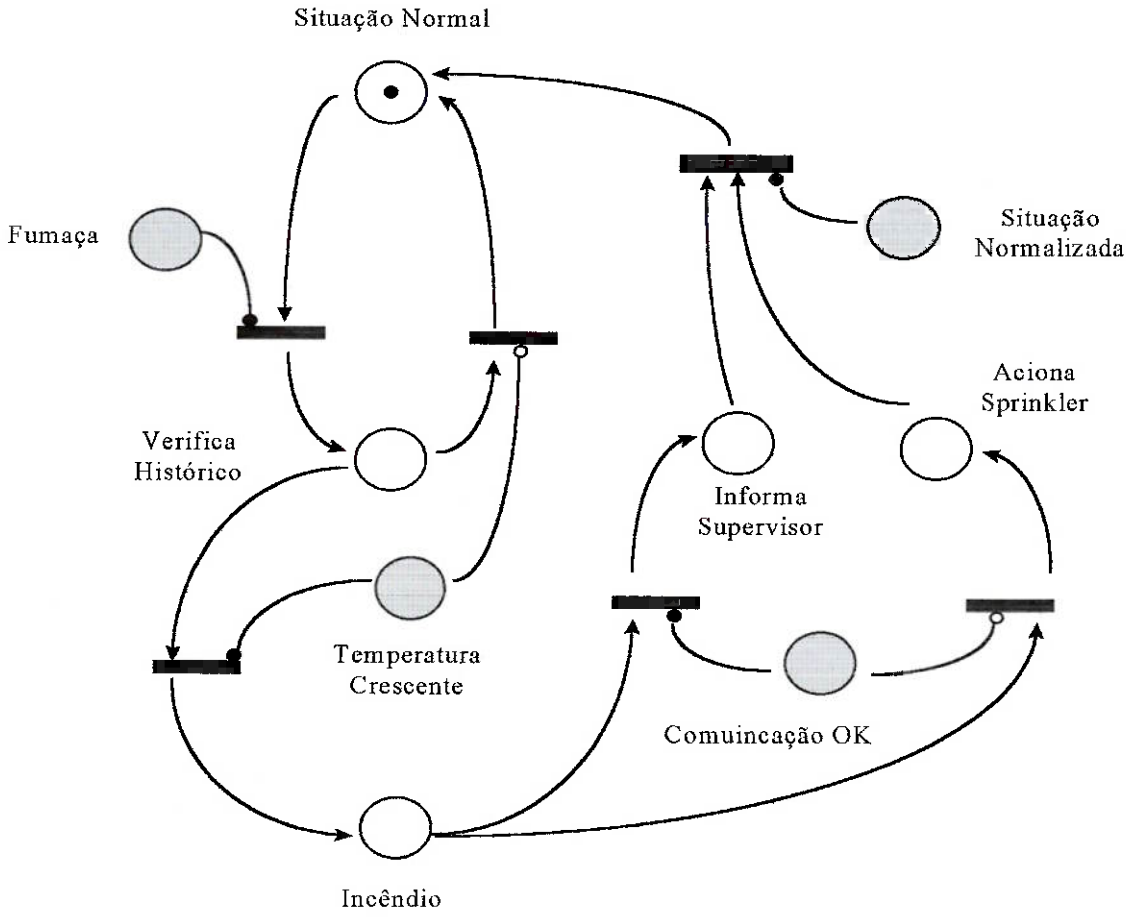


Figura 5.2 - Rede de detecção de incêndio

Apêndice 5.1.3 Alarme

O alarme de incêndio deve ser disparado por um comando vindo do supervisório de nível superior. A função do alarme é avisar uma situação de emergência. A rede correspondente ao acionamento do alarme é mostrada em 5.3.

Variáveis de entrada: não se aplicam.

variáveis de saída: disparo do alarme.

Variáveis de informação: não se aplicam.

variáveis de comando: situação de emergência.

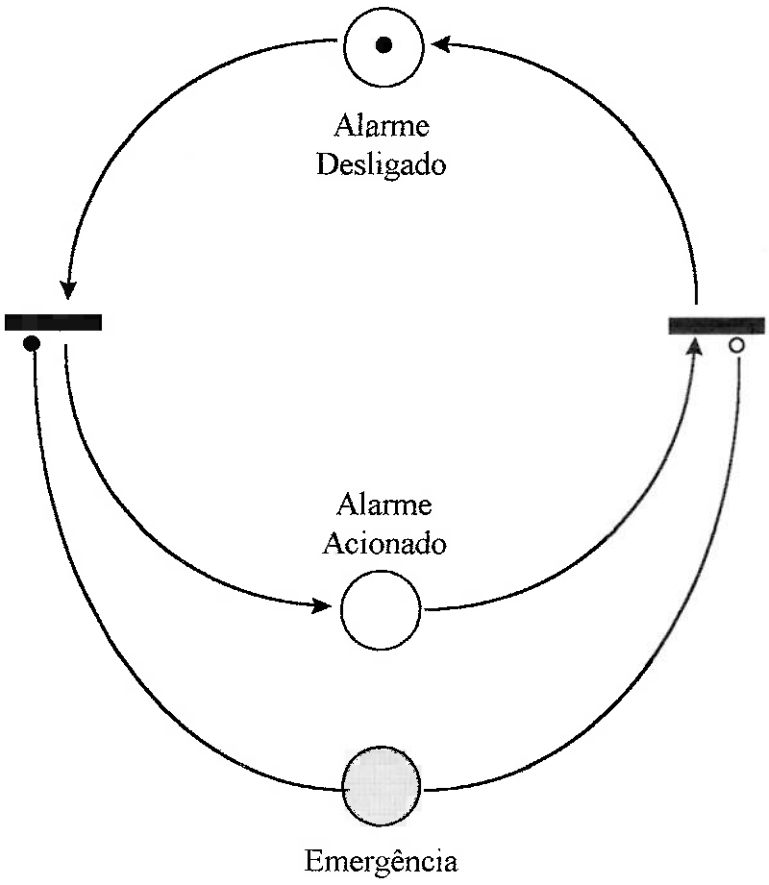


Figura 5.3 - Rede de acionamento do alarme

5.2. *Sistema de iluminação*

O sistema de iluminação deverá ligar e desligar a iluminação da sala, conforme a sala esteja ou não habilitada e de acordo com o seu estado (habilitada/desabilitada). A rede que representa seu comportamento é mostrada em 5.4

variáveis de entrada: presença.

variáveis de saída: ligar iluminação.

variáveis de informação: não se aplicam.

variáveis de comando: habilitação da sala.

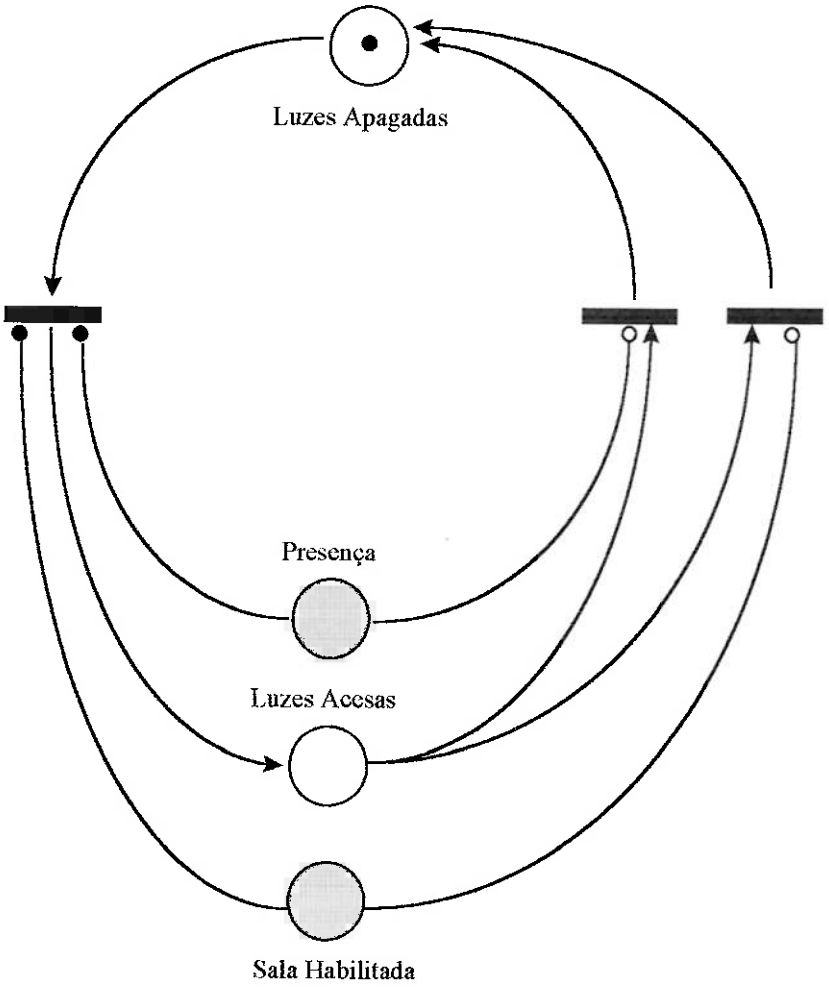


Figura 5.4 - Rede de acionamento da iluminação

5.3. *Sistema de ventilação*

O sistema de ventilação deve ajustar a velocidade do ventilador de acordo com a temperatura atual da sala e a temperatura ajustada pelo usuário. Além disso, deve levar em conta os sinais de presença na sala e de habilitação da sala.

Dois importantes aspectos devem ser salientados nessa rede. O primeiro diz respeito à natureza contínua da velocidade do ventilador. Essa variável foi discretizada e, ao se ajustar a velocidade do ventilador, ela é incrementada ou decrementada de um valor

fixo conforme o caso. O outro aspecto relevante é a necessidade de se colocar um atraso após se fazer um ajuste na velocidade. Isso ocorre devido ao fato de o sistema térmico ter constante de tempo alta. Caso não se adote essa precaução, o ventilador passará rapidamente do estado totalmente ON para o totalmente OFF, se tornando equivalente a um controle ON-OFF.

5.3.1. Atuação na velocidade do ventilador

A rede que determina que atitude se tomar com relação à velocidade do ventilador dadas a temperatura na sala, a temperatura de ajuste, a presença ou não de pessoas na sala e a habilitação ou não da sala é mostrada na figura 5.5.

variáveis de entrada: temperatura na sala e presença.

variáveis de saída: aumento/diminuição da velocidade do ventilador.

variáveis de informação: não se aplicam.

variáveis de comando: habilitação da sala.

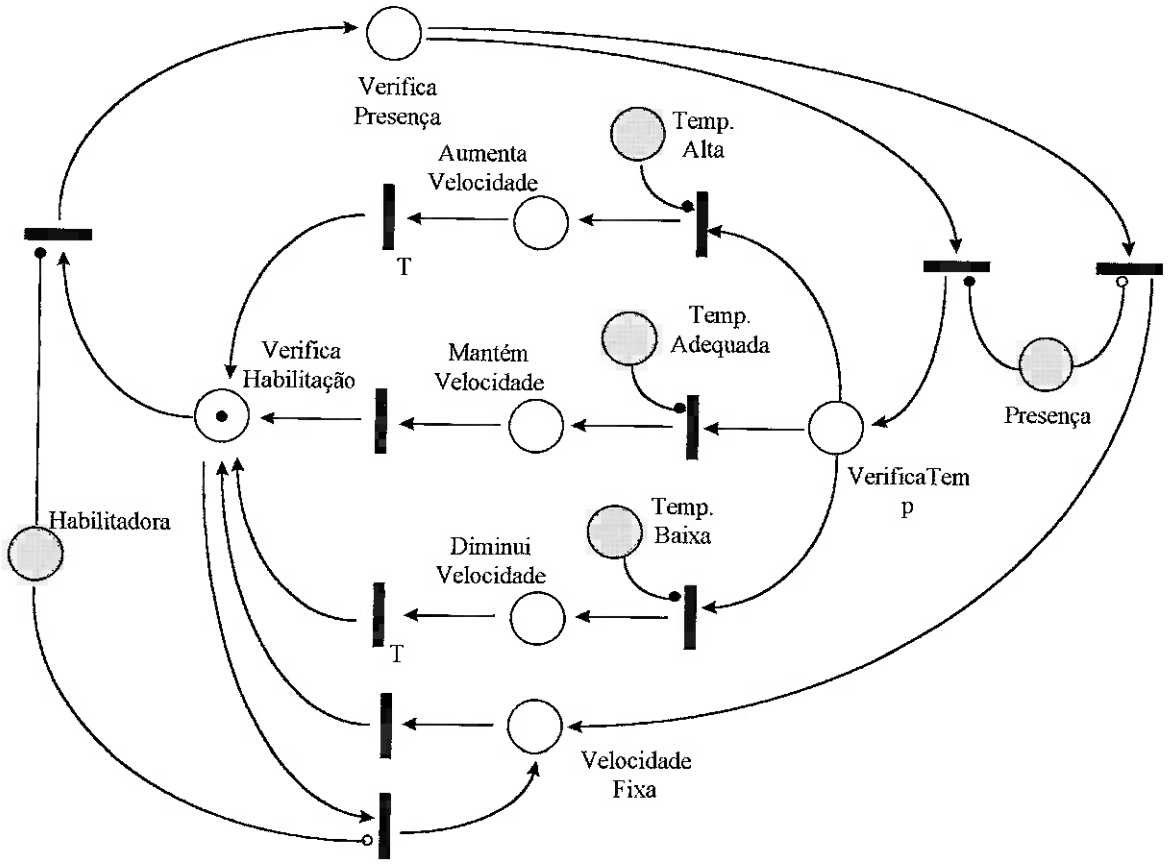


Figura 5.5 - Rede de ajuste da velocidade do ventilador

5.3.2. Informação da velocidade do ventilador

variáveis de entrada: temperatura na sala e presença.

variáveis de saída: não se aplicam.

variáveis de informação: velocidade do ventilador.

variáveis de comando: não se aplicam.

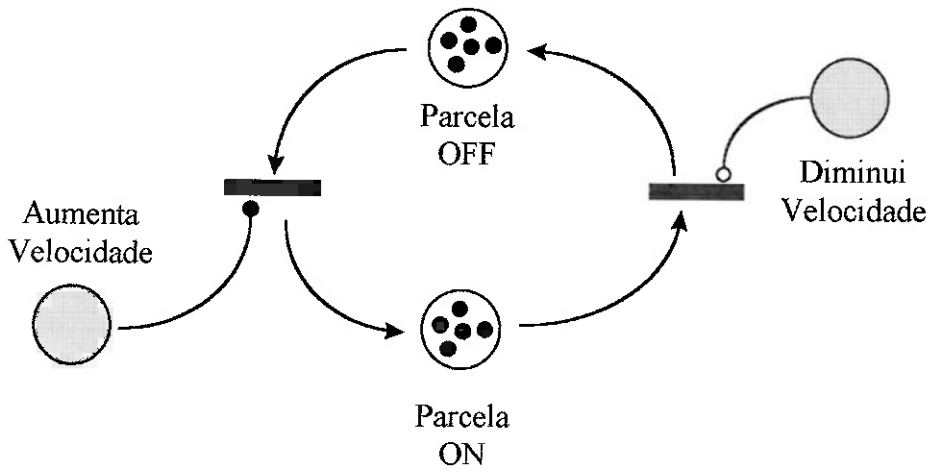


Figura 5.6 - Rede de atualização da velocidade do ventilador

6. Integração entre os subsistemas

No capítulo anterior obtivemos os modelos de cada subsistema separadamente. Mas este trabalho visa justamente à integração destes subsistemas. Devemos então modelar o funcionamento conjunto destes sistemas. Com isso podemos avaliar a interação entre os subsistemas e analisar os pontos em que eles se inter-relacionam.

Primeiramente, vê-se o sistema de detecção de intrusão (figura 6.1). Podemos notar que este sistema se utilizada da mesma variável utilizada no sistema de iluminação (**Sala Habilitada**). O funcionamento desta rede é bastante simples, não requerendo um maior aprofundamento em seu estudo.

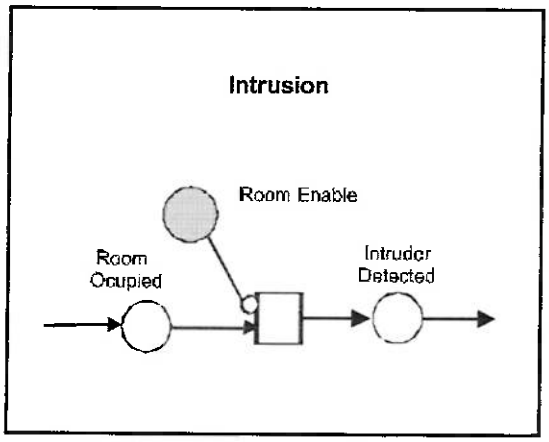


Figura 6.1 - Sistema de Intrusão

O funcionamento é bastante simples: se um sinal de presença é verificado em uma sala desabilitada, um sinal de intrusão será enviado ao supervisório.

O sistema de iluminação é bastante simples também. Com a sala estando habilitada, o sinal de presença ocasiona o acendimento das lâmpadas da sala. Este sistema é mostrado na figura 6.2.

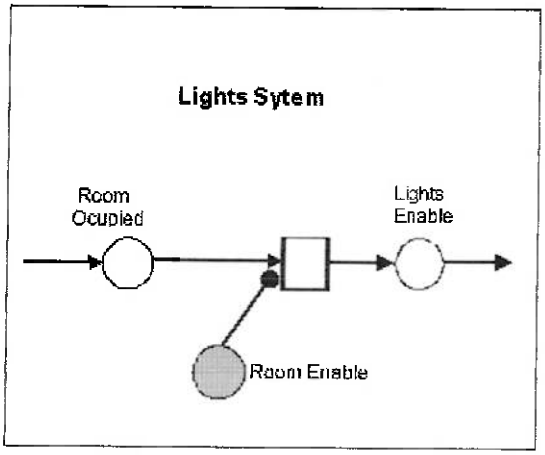


Figura 6.2 - Sistema de Iluminação

Pode-se agora agrupar os sistemas de iluminação e detecção de intrusão para quando a sala estiver ocupada, como pode-se observar na figura 6.3.

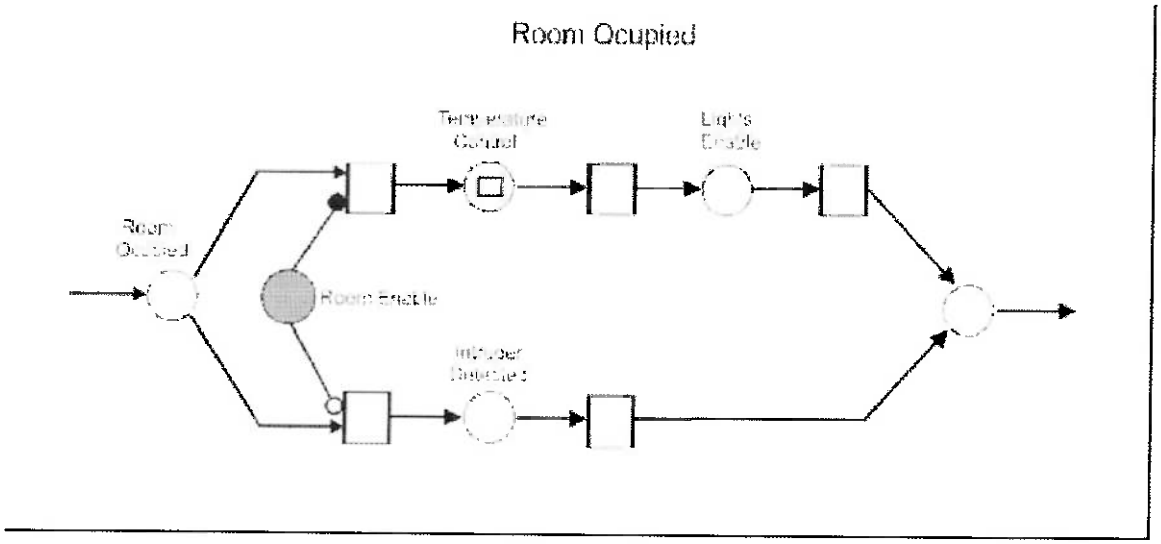


Figura 6.3 - Sistema de Iluminação

Os sistemas de alarme e detecção de incêndio se inter-relacionam intimamente, e portanto a sua modelagem conjunta se mostra uma estratégia bastante sensata e que permitirá uma melhor compreensão do comportamento do sistema ante eventos de incêndio e do disparo do alarme para incêndio.

Outro detalhe que não deve ser esquecido na modelagem destes sistemas é que ele deve ser capaz de resetar os estados de acionar alarme e de incêndio.

A rede abaixo mostra o comportamento conjunto destes sistemas. Quando um sinal de fumaça é detectado na sala, juntamente com um sinal de calor excessivo, pressupõe-se que há incêndio na sala.

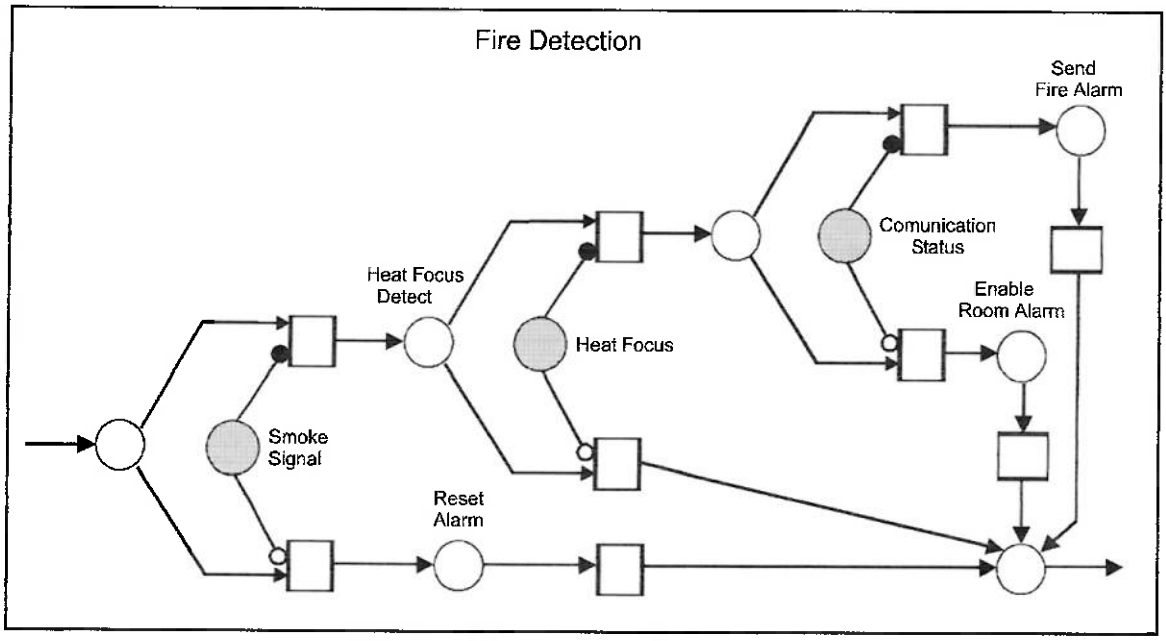


Figura 6.4 - Sistemas de detecção de incêndio e alarme de incêndio

O comportamento do sistema é distinto caso a comunicação do CLP com o supervisório esteja funcionando ou não:

- Com a comunicação OK, um sinal de incêndio será enviado ao supervisório, que tratará de tomar as medidas adequadas.
- Com a comunicação cortada, o CLP disparará o alarme da sala.

Com a comunicação OK, a resposta do sistema pode ser bem mais abrangente e inteligente. Caso o supervisório note que existe incêndio, as luzes de emergência e alarmes do prédio podem ser acionados, o Corpo de Bombeiros acionado, os elevadores

desligados, etc. No entanto, o CLP por si só pode responder a esta situação mesmo sem o supervisório, obviamente com uma ação limitada.

O sistema de ventilação compõe-se de dois sub-sistemas: atuação sobre a velocidade do ventilador e a informação da velocidade do ventilador. Esses dois sub-sistemas foram implementados em uma mesma rede, de forma a facilitar seu estudo conjunto e verificar a interferência de um em outro.

O funcionamento desta rede prevê de início a leitura da temperatura da sala. A seguir faz-se a comparação desta temperatura com o *setpoint* pré-estabelecido. Caso a temperatura esteja dentro da faixa adequada, a velocidade do motor se mantém constante. Se a temperatura estiver mais alta do que a desejada, a velocidade do ventilador será aumentada, e se temperatura estiver abaixo da especificada, então dimunuir-se-á a velocidade do ventilador.

Para completar a modelagem do sistema, deve-se lembrar de considerar como o sistema age quando a sala está desocupada.

Neste caso as luzes são mantidas desligadas, e a velocidade do ventilador é mantida constante em um valor pré-estabelecido.

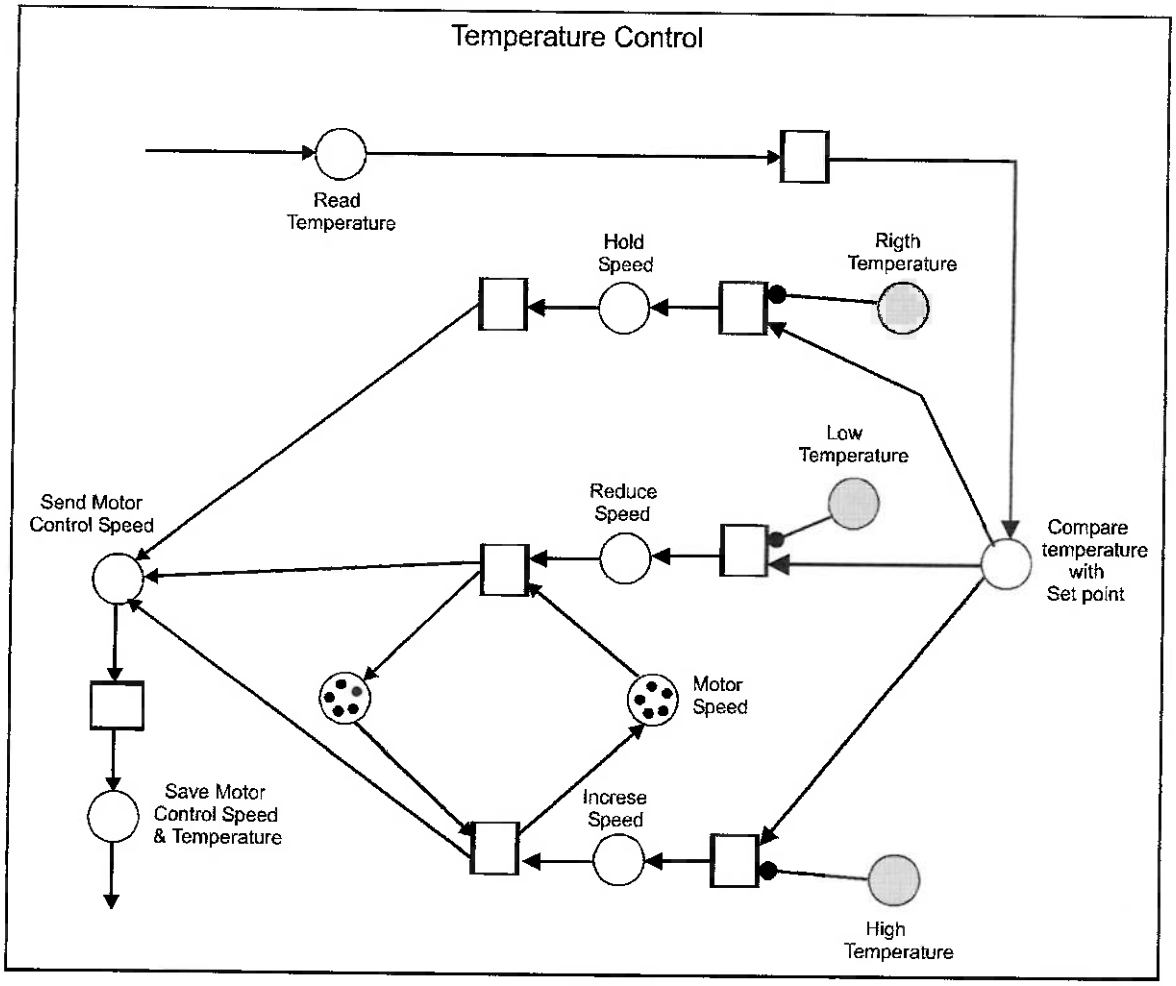


Figura 6.5 - Sistema de Ventilação

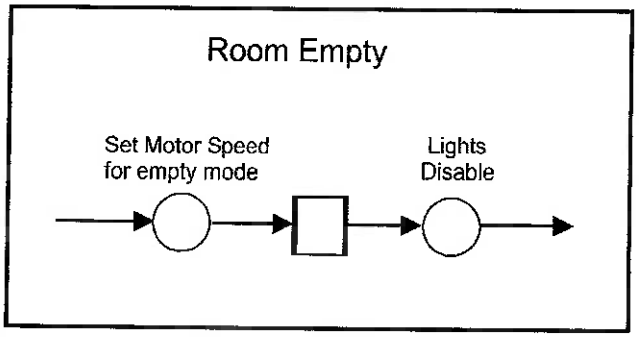


Figura 6.6 - Sala desocupada

O sistema supervisório pode ser visualizado como uma rede, como visto na figura abaixo:

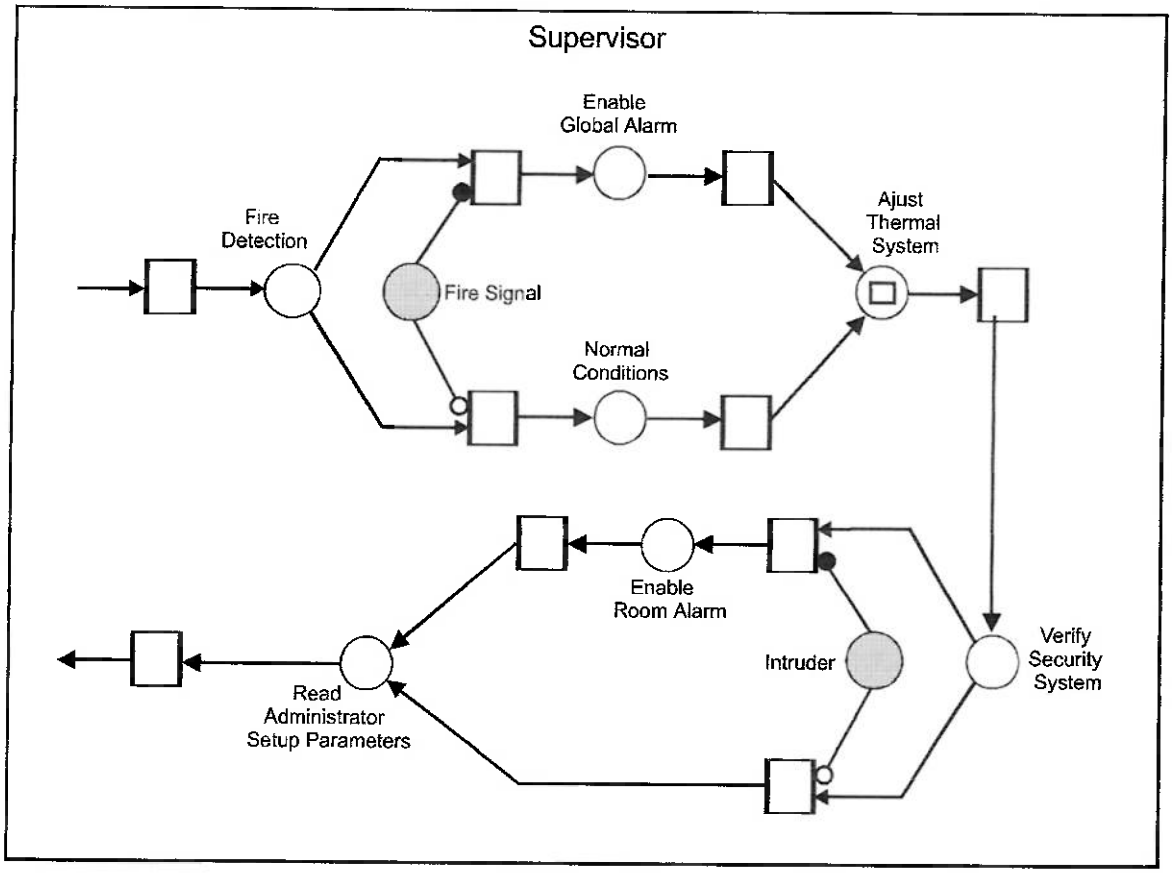


Figura 6.7 - Rede representativa do sistema supervisório

Caso o supervisório receba um sinal de incêndio em alguma das salas, ele habilita o alarme nas salas, e ajusta o sistema térmico. O supervisório pode também decidir que condições normais foram reestabelecidas, e desabilitar os alarmes.

O supervisório recebe também o sinal de intrusão. Neste caso também pode acionar os alarmes das salas.

O módulo do CLP pode ser modelado pela seguinte rede:

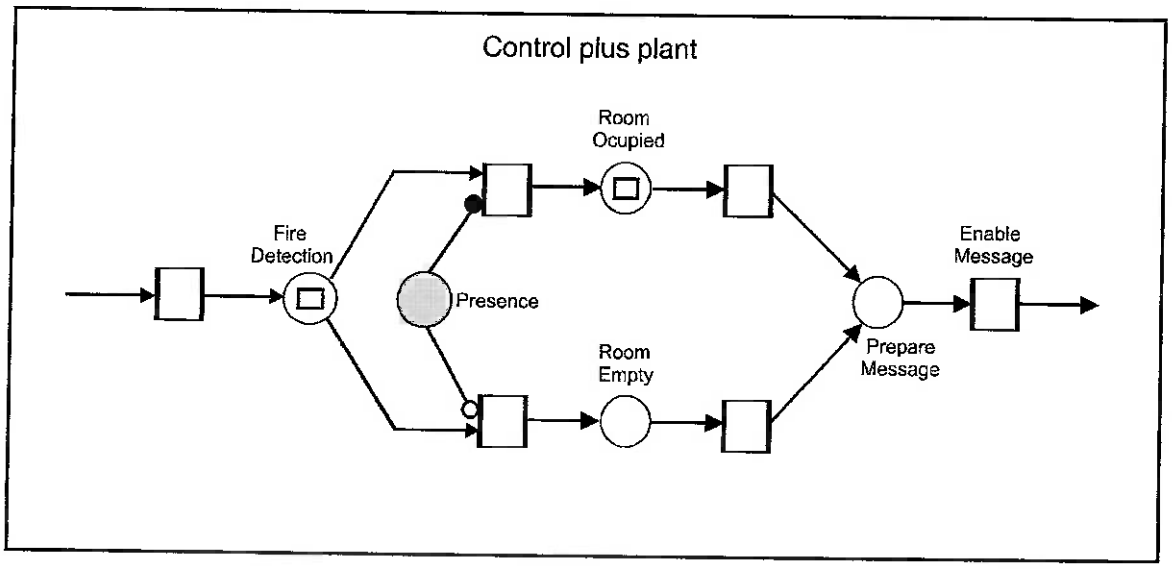


Figura 6.8 – Modelo do programa de controle do CLP

O programa no CLP faz a verificação de incêndio e de presença na sala. A seguir envia estes dados ao supervisório (se a comunicação entre ambos estiver funcionando).

O sistema como um todo pode ser visualizado na figura abaixo:

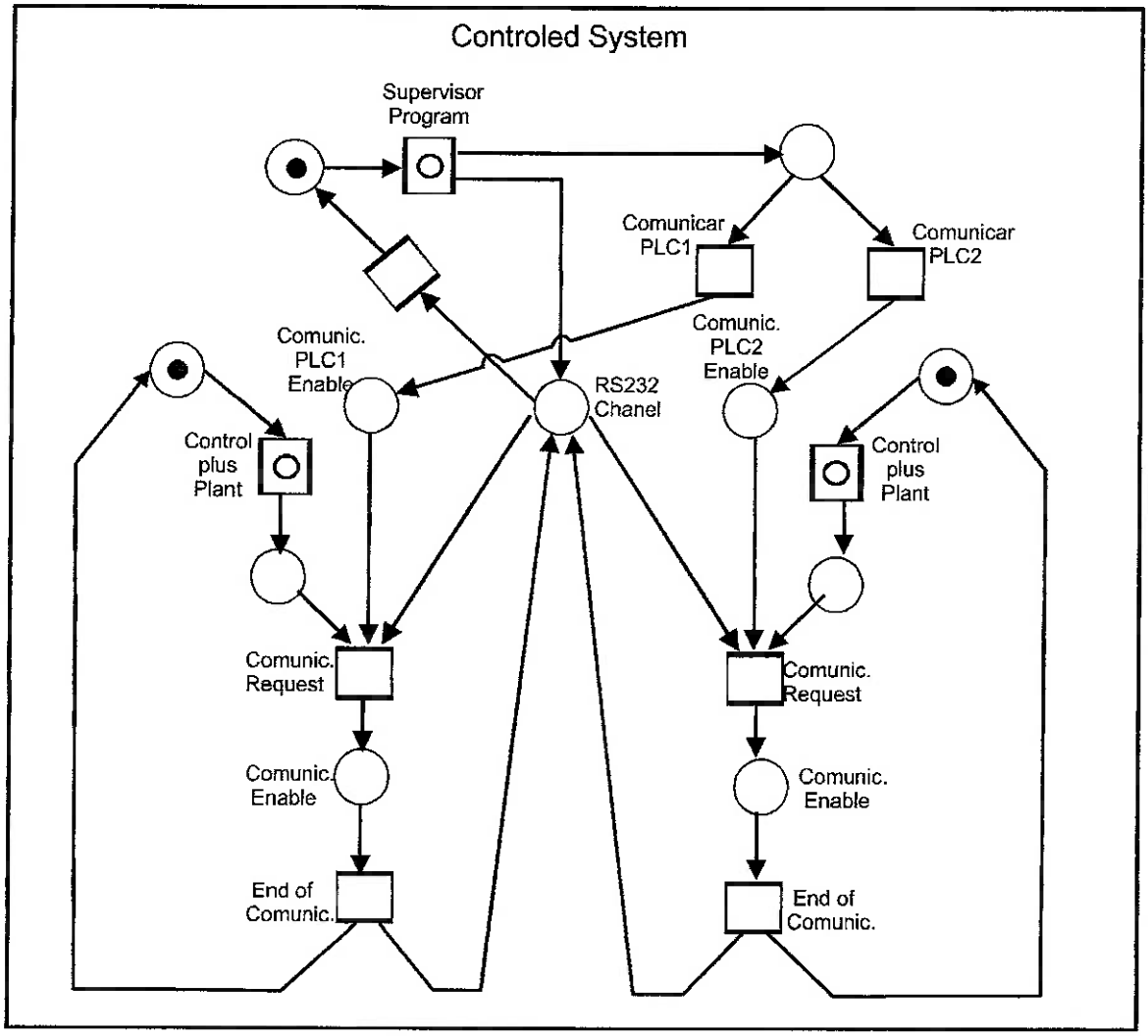


Figura 6.9 - Sistema CLP-Supervisório

Nota-se que cada CLP se comunica com o supervisório, enviando e recebendo sinais através da RS232C.

7. Implementação

O ambiente onde o sistema de controle será observado é uma maquete, constituída de duas salas. Cada uma das salas da maquete terá um CLP ligado a vários sensores, a saber:

- sensor de presença;
- sensor de fumaça;
- sensor de temperatura;

Além disso em cada sala haverá uma lâmpada e um alarme. Uma das salas estará equipada com um ventilador para refrigeração.

Para o acionamento da lâmpada e do alarme, haverá uma saída do CLP ligada a um circuito de potência (basicamente um relé, de forma a garantir a potência mínima para o acionamento).

O acionamento do ventilador também deve ter um circuito de acionamento associado. Esse acionamento pode ser feito de forma mais eficiente por meio de um circuito capaz de realizar a modulação de voltagem, conhecido como PWM (*Pulse Width Modulator*). No Anexo II este circuito será apresentado com maiores detalhes.

Os dois CLPs serão ligados a um único computador PC que fará o papel de sistema supervisório. Portanto, para a implementação do programa de controle, será necessário que haja uma interface de comunicação entre os CLPs e o computador.

No sistema supervisório também haverá um banco de dados, onde podem-se armazenar os dados pertinentes, que poderão ser acessados por um supervisório de ordem

superior. A discussão sobre que dados devem ser disponibilizados é assunto de grande complexidade e que portanto foge ao escopo do trabalho.

A seguir é mostrado um esquema da maquete:

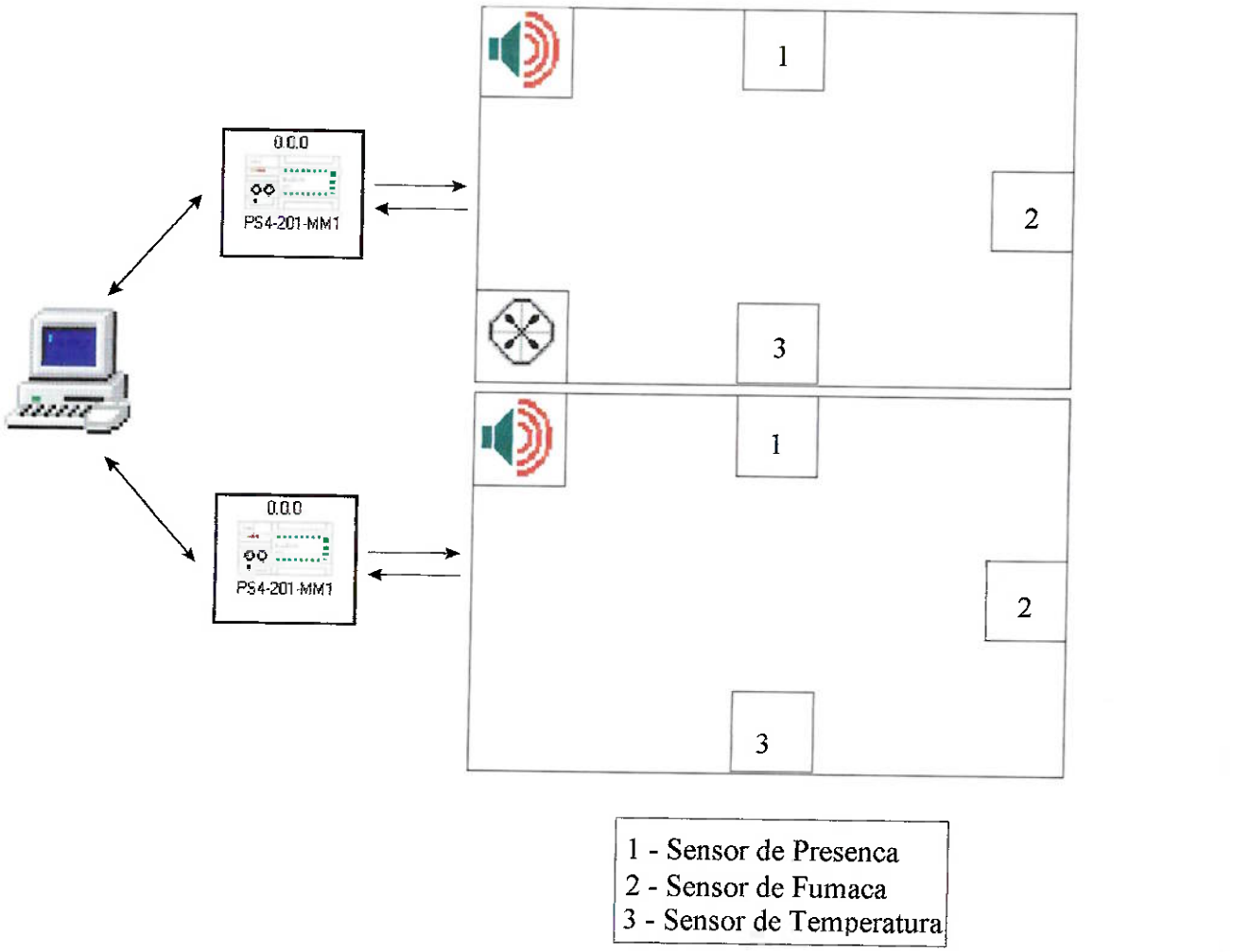


Figura 7.1 - Esquema da Maquete a ser Implementada

8. Desenvolvimento do Software de Controle

A seguir, se encontram descritos os componentes do software de controle e a forma como eles se interrelacionam:

8.1. Componentes do software de controle:

O software responsável pelo controle do ambiente se encontra dividido em três módulos:

- módulo de controle: o software armazenado no CLP, responsável pela atuação e sensoreamento no sistema;
- módulo de comunicação: O software de comunicação *DDE Server*, fornecido pela Klockner-Moeller, fabricante do CLP;
- módulo supervisório: armazenado em um microcomputador do tipo PC, responsável pelas atividades de monitoração do sistema e ajuste de parâmetros visando maior eficiência.

A interrelação entre os três programas pode ser vista na figura 8.1. Nela, observa-se que o software *DDE Server* atua como uma ponte entre o sistema supervisor e o software de controle armazenado no CLP.

É importante se salientar que o módulo armazenado no CLP é capaz de atuar de forma autônoma, pois as atividades de controle mais básicas cabem exclusivamente a esse sistema. O supervisório tem apenas atividades de coordenação e monitoração do sistema, visando aumentar a sua eficiência e a relaização de ajustes. Com isso, não se pode admitir que a interrupção da comunicação entre esses dois módulos leve a um colapso do sistema

de controle. Isso é especialmente verdade em situações de emergência, como por exemplo na ocorrência de um incêndio. Nessas situações, a comunicação pode estar severamente danificada, sendo portanto de absoluta importância que o nível mais baixo atue no sentido de evitar danos maiores.

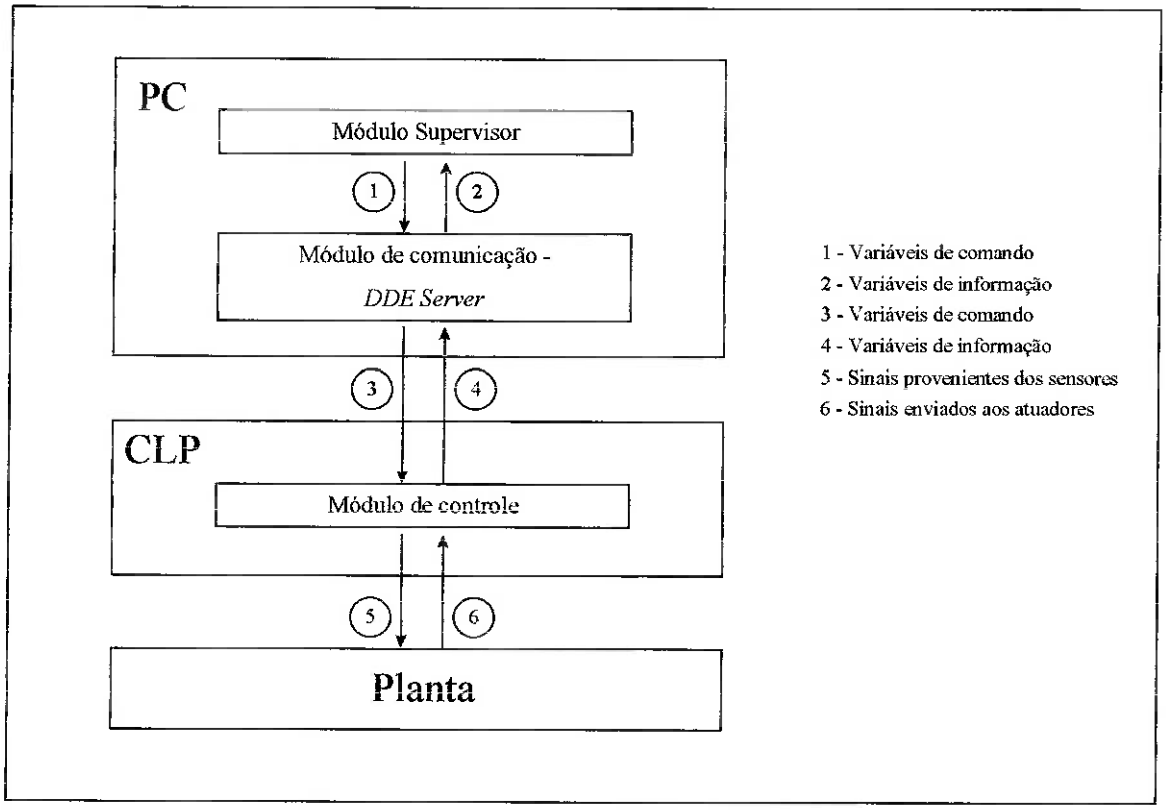


Figura 8.1 - Relação entre os módulos de software do sistema de controle.

Por outro lado, cabe ressaltar que o fato de o sistema instalado no CLP poder agir de forma independente não significa que o funcionamento do sistema seja o mesmo nos momentos em que a comunicação está presente e aqueles em que ela se encontra comprometida. Conforme se verifica nos modelos desenvolvidos para a descrição do funcionamento do sistema de controle, o fato de a comunicação estar presente permite que

o sistema atue de forma mais coordenada, sendo capaz de interpretar dos dados coletados com maior acurácia e, consequentemente, agindo de forma mais eficiente.

A seguir, segue uma breve descrição de cada um dos componentes de software envolvidos:

8.1.1. DDE Server:

Esse módulo é fornecido pelo próprio fabricante do CLP e é responsável por estabelecer a comunicação entre os dois outros módulos de software envolvidos no sistema de controle.

Essa comunicação é feita através do sistema de DDE (*Dynamic Data Exchange*) componente do Windows. Esse sistema permite que duas aplicações troquem dados entre si durante a execução. O *DDE Server* atua como servidor de dados, a partir do qual o sistema supervisório coleta os dados referentes ao estado do CLP. Além disso, é possível se enviar dados do supervisório para o CLP por meio do *DDE Server*.

8.1.2. Módulo Supervisório:

Esse módulo do sistema de controle foi desenvolvido utilizando-se o LabWindows, um ambiente de programação visual para Windows, desenvolvido pela National Instruments que faz uso da linguagem C de programação. Seu uso se mostra vantajoso devido à grande quantidade de controles previamente definidos e orientados em grande parte ao desenvolvimento de softwares de controle.

Parte da função desse módulo consiste em fazer a interface entre o usuário e o sistema de controle. É esse módulo que apresenta o estado atual das salas ao usuário e que permite que os ajustes desejados sejam realizados. Para isso, realiza leituras periódicas

do estado do CLP através do DDE Server e, depois de processar esses dados juntamente com as entradas fornecidas pelo usuário, envia sinais ao CLP.

Além disso, esse módulo permite que se simule os efeitos das entradas provenientes de níveis superiores de controle. No sistema desenvolvido, essas entradas são fornecidas pelo próprio usuário, visando verificar os efeitos das decisões desses níveis de controle sobre o funcionamento do sistema.

Uma outra importante função desse módulo é gerar um sinal enviado ao CLP que permite monitorar o estado da comunicação entre esses dois módulos. A forma como essa verificação é feita é descrita em detalhes em 8.3.

A descriçao da forma de utilização e apresentação das telas desse componente do software é feita no Anexo III.

Atualmente, esse módulo está sendo executado em um microcomputador do tipo PC dentro do ambiente Windows95.

8.1.3. Módulo de controle (CLP):

A parte do software que é executada no CLP é a responsável por concretizar todas as decisões de controle tomadas através do acionamento dos atuadores. Além disso, cabe a esse módulo coletar todos os dados referentes ao estado da planta que serão usados na tomada de decisões de controle.

Em situações em que a comunicação com o supervisório está rompida, esse módulo é capaz de manter o sistema em funcionamento, ainda que fora de seu ponto ótimo. Em momentos de normalidade da comunicação com o supervisório, esse módulo

combina as informações provenientes da planta com os comandos enviados pelo supervisório de forma a atuar de forma mais eficiente.

O envio de dados ao supervisório é feito por meio de variáveis que são lidas pelo DDE Server e disponibilizadas para o sistema supervisório. A recepção de dados é feita de maneira semelhante, com o DDE Server escrevendo em veriáveis do CLP os valores enviados pelo sistema supervisório.

Esse módulo conta também com um algoritmo que permite identificar o estado da comunicação com o sistema supervisório. A forma como isso é feito está descrita em detalhes em 8.3.

8.2. Estrutura dos módulos supervisório e de controle:

Como já foi descrito acima, o sistema é composto por três módulos, sendo que dois deles se dedicam a atividades de controle propriamente ditas e o terceiro se presta a fazer a comunicação entre eles. Uma vez que o módulo de comunicação é fornecido pelo fabricante do CLP, pouco pode ser dito a respeito de sua estrutura de funcionamento. Entretanto, o mesmo não é verdade para o módulo supervisório e o módulo de controle (armazenado no CLP). A seguir, descreve-se o funcionamento desses dois componentes de software.

O princípio de funcionamento dos dois módulos é semelhante: o programa realiza repetidos ciclos e, a cada um desses ciclos repete a seguinte rotina:

- Leitura de dados e variáveis;
- Tratamento dos dados e variáveis visando a tomada de decisões;
- Execução das decisões tomadas.

No software supervisório, o trecho de leitura de dados e variáveis consiste em verificar as entradas fornecidas pelo usuário (que aqui atua também no papel de supervisório de ordem superior) e obter o estado das variáveis enviadas pelo CLP. No ciclo de tratamento dos dados é feito todo o processamento envolvendo os dados adquiridos na primeira parte do ciclo, como por exemplo *setar* a variável de acionamento dos sprinklers a partir dos dados enviados pelo CLP e da resposta dada pelo usuário a esse evento. No momento da atuação, essa variável será enviada ao CLP para que esse acione os sprinklers. Nesse trecho do ciclo, também se faz a atualização dos dados referentes ao estado da sala que são mostrados ao usuário.

No software de controle armazenado no CLP, a leitura dos dados corresponde à aquisição dos sinais enviados pelos sensores e a leitura das variáveis enviadas pelo supervisório. O tratamento dessas entradas consiste em se determinar quais atuadores devem ser acionados e de que forma. Além disso, as variáveis de comunicação com o supervisório são ajustadas de acordo com a situação da sala. O trecho de atuação consiste no envio de sinais aos atuadores de forma a executar as ações de controle do sistema.

É importante se salientar que no programa elaborado para o CLP, esses três momentos do programa não se apresentam de forma explícita, pois muitas das variáveis de entradas e saída já se encontram vinculadas aos respectivos endereços de entrada e saída, não sendo necessário portanto realizar comandos para a leitura ou escrita dessas variáveis. Entretanto, a cada ciclo do programa, o sistema do CLP se encarrega de atualizar as variáveis de entrada e saída, correspondendo esse momento do programa aos ciclos de leitura e execução.

Uma grande diferença entre os módulos supervisório e de controle diz respeito aos períodos em que os ciclos são executados.

No CLP, a freqüência (e portanto o período) de execução do ciclo de controle é ditada pelo tamanho do programa. Para as ações que devam ser tomadas em intervalos bem definidos, é necessário se fazer o uso de temporizadores. Apesar de o período de execução estar vinculado ao tamanho do programe, existe um limite no tempo que pode ser gasto em cada ciclo de execução. No caso do CLP usado - o PS4-201-MM1 da Klockner Moeller, esse limite é de 60ms/ciclo.

No sistema supervisório, o período pode ser definido utilizando-se funções de temporização. O LabWindows permite que se defina uma função que é chamada em intevalos ajustáveis de tempo. O próprio Windows permite que se execute um conjunto de procedimentos em intervalos de tempo definidos por meio da função *SetTimer*. Apesar dessas facilidades, é necessário garantir que o tempo de processamento gasto em um ciclo esteja abaixo do período ajustado para a chamada da função.

Outra grande diferença entre os dois módulos diz respeito ao tipo de atitude tomada. Enquanto o supervisório é responsável apenas pelo ajuste de parâmetros de execução e pela monitoração do sistema (atividades lógicas), o módulo de controle presente no CLP deve realizar tanto atividades de processamento de dados (lógicas) como o acionamento de dispositivos e recepção de sinais elétricos provenientes de sensores (atividades de *hardware* - físicas).

8.3. Comunicação entre os módulos:

Conforme já descrito em 8.1.1, a comunicação entre o sistema supervisório e o programa em execução no CLP é feito por meio de DDE utilizando o *software DDE Server* fornecido pelo próprio fabricante do CLP.

Um problema presente na comunicação entre os módulos consiste em como um dos módulos pode verificar se o outro módulo está ou não presente. Esse problema foi solucionado de formas distintas no módulo de controle (CLP) e no supervisor.

8.3.1. Verificação da comunicação pelo supervisor:

Para verificar a comunicação com o CLP, o supervisor faz uso de uma propriedade da comunicação DDE: o DDE Timeout. Cada vez que o supervisor envia ou obtém dados do CLP (via DDE Server), ele aguarda uma resposta por um determinado intervalo de tempo. Caso essa resposta não venha dentro desse intervalo, admiti-se que há problemas na comunicação. A comunicação DDE entre esse módulo e o DDE Server é encerrada e o usuário é notificado sobre esse distúrbio. Depois disso, encerra-se o módulo DDE Server, pois, em casos onde a ligação entre o CLP e o microcomputador é rompida, esse módulo pode causar lentidão no sistema.

O intervalo de tempo para que se considere que houve falha no sistema é ajustável, tendo sido usado um intervalo de 5s no programa implementado. Esse intervalo relativamente grande foi escolhido de forma que o supervisor não encarasse qualquer atraso como uma falha de comunicação.

8.3.2. Verificação da comunicação pelo módulo de controle (CLP):

Para que o módulo de controle possa comprovar a comunicação adotou-se um algoritmo no qual o supervisor envia ao CLP uma "onda quadrada" de período bem definido. Esse efeito de onda quadrada é gerado fazendo-se com que o supervisor alterne o valor de uma variável a cada meio período e a escreva em um registrador do CLP. O CLP faz então uma verificação periódica do valor desse registrador. Caso esse valor se mantenha constante ou seja diferente dos dois níveis da onda, percebe-se que há uma falha de comunicação.

Para se garantir que esse algoritmo de verificação de comunicação funcione é necessário se garantir uma compatibilidade entre o período da "onda quadrada" gerada pelo supervisor e o período da amostragem da variável feita pelo módulo de controle.

O meio período da onda quadrada será adotado como sendo o período do ciclo do programa supervisor, ou seja, a cada vez que o supervisor executa o ciclo descrito em 8.2, ele altera o valor da variável de comprovação de comunicação e a envia ao CLP. Esse é o período que torna a verificação da comunicação mais rápida, pois é a maior freqüência com que o supervisor pode alternar o valor da variável seguindo seu ciclo de funcionamento.

A freqüência de amostragem da variável executada pelo CLP deve ser maior que a freqüência da onda gerada pelo supervisor. Caso essa freqüência de amostragem esteja acima de duas vezes a freqüência da onda em questão, garante-se que todas as bordas, sejam elas de descida ou de subida, serão detectadas pelo módulo de controle.

Naturalmente, o CLP não detectará bordas em todas as amostragens feitas e, portanto, a variável de detecção de borda por si só não pode ser usada como a variável de

estado de comunicação. A variável de estado de comunicação é atualizada em intervalos constantes de tempo por meio do uso de um temporizador. A atualização dessa variável deve ser feita em intervalos suficientemente grandes de forma a permitir que o supervisor tenha tempo de alterar o valor da variável, pois, em caso contrário, será detectado um ciclo em que não houve variação e isso não terá ocorrido por falha na comunicação, mas sim pelo fato de o supervisor não ter tido tempo de gerar uma borda para detecção. Como a onda gerada passa metade do ciclo em um dos valores e a outra metade do ciclo no outro valor, basta garantir que a freqüência de atualização da variável de comunicação seja inferior a duas vezes a freqüência da onda quadrada gerada.

É importante notar que, devido ao fato de a variável de estado de comunicação ser atualizada em intervalos regulares, surge uma discretização no tempo no que se refere a essa variável, pois, caso a comunicação se reestabeleça, o estado da variável de comunicação só será atualizado no ciclo seguinte.

Um importante fator a ser levado em conta é o atraso que pode ocorrer na comunicação dessas variáveis de controle de comunicação.

Em casos onde o atraso na comunicação é constante, o algoritmo poderá apresentar atrasos decorrentes da demora na transmissão da onda, ou seja, o estado da comunicação se altera, mas esse fato só será percebido depois de um intervalo de tempo igual ao atraso envolvido. Isso poderá causar problemas em casos onde esses atrasos sejam muito grandes, principalmente o estado da comunicação for constantemente alterado, uma vez que, por um intervalo de tempo igual ao atraso, a variável de estado de tempo estará desatualizada.

Em casos onde o atraso de transmissão possa variar, é necessário que ajustes entre as freqüências envolvidas sejam feitos de forma a comportar os efeitos desse atraso. Considerando que os próprios atrasos podem ser encarados como falhas de comunicação, esses ajustes dependerão de quão tolerante se deseja ser com os atrasos de transmissão dos sinais.

Anexo A. Descrição do Módulo Supervisório

O programa correspondente ao módulo supervisório é composto de duas telas principais e uma terceira que permite a visualização do histórico de temperatura e velocidade do motor em cada uma das salas.

A seguir, encontra-se uma breve descrição de cada uma dessas telas e os controles encontrados em cada uma delas.

A tela *User Input* possui controles que permitem que o usuário ajuste os parâmetros de funcionamento de cada uma das salas. Nessa janela, existem duas seções, cada uma correspondendo a uma das salas e contendo os mesmos controles que são descritos abaixo:

Setpoit Temperature: É um potênciometro que se ajuste a temperatura de conforto da sala entre 20°C e 26°C. Atualmente, esse controle se encontra desabilitado para a sala 2, pois essa ainda não está equipada com um sistema de ventilação.Note-se que essa será a temperatura de conforto da sala apenas nos momentos em que há comunicação entre o CLP e o supervisório, pois, em caso contrário, essa temperatura é ajustada no próprio CLP, por meio de um potenciômetro.

IncCtrlSpeed: Permite que se ajuste o incremento de velocidade do ventilador. Esse incremento corresponde a variação que será feita na velocidade do ventilador ao se notar que ela deve ser ajustada. Da mesma forma que o potenciômetro de ajuste de temperatura, está disponível apenas para a sala 1 no momento.

Anexo A.1 Tela Setup Parameters:

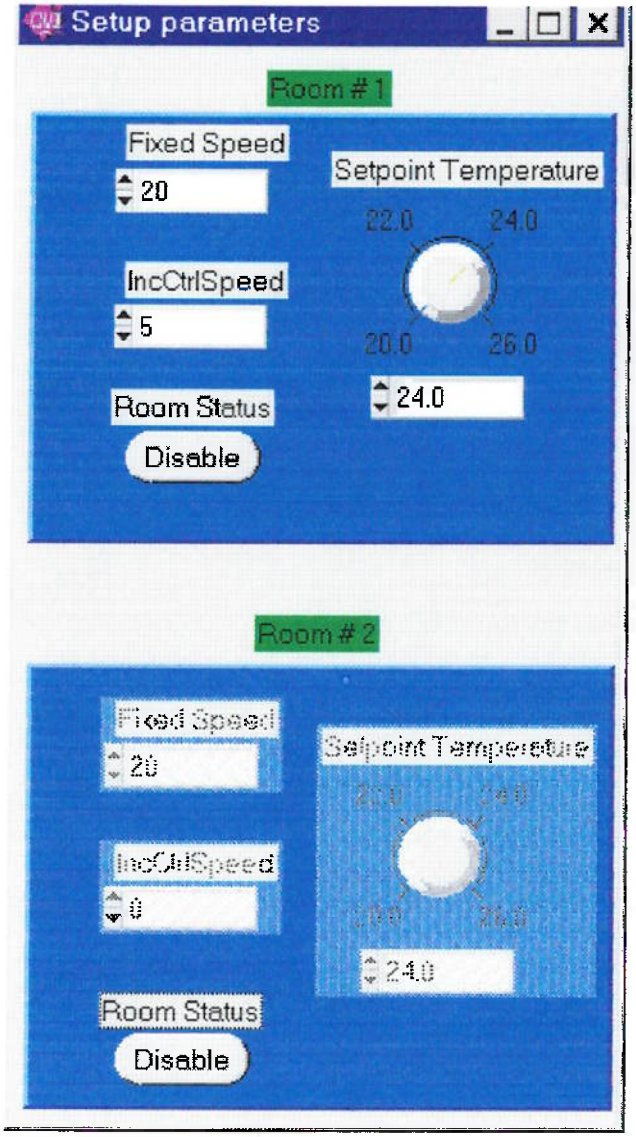


Figura A.1 - Tela do *Setup Parameters*

Fixed Speed: Permite que se ajuste a velocidade em que o ventilador funcionará quando a sala estiver desabilitada ou vazia.

Room Status: Permite que se altere o estado de cada sala entre habilitada e desabilitada. Esse comando terá efeitos no controle de intrusão e de iluminação da sala.

Anexo A.2 Tela Control Panel:

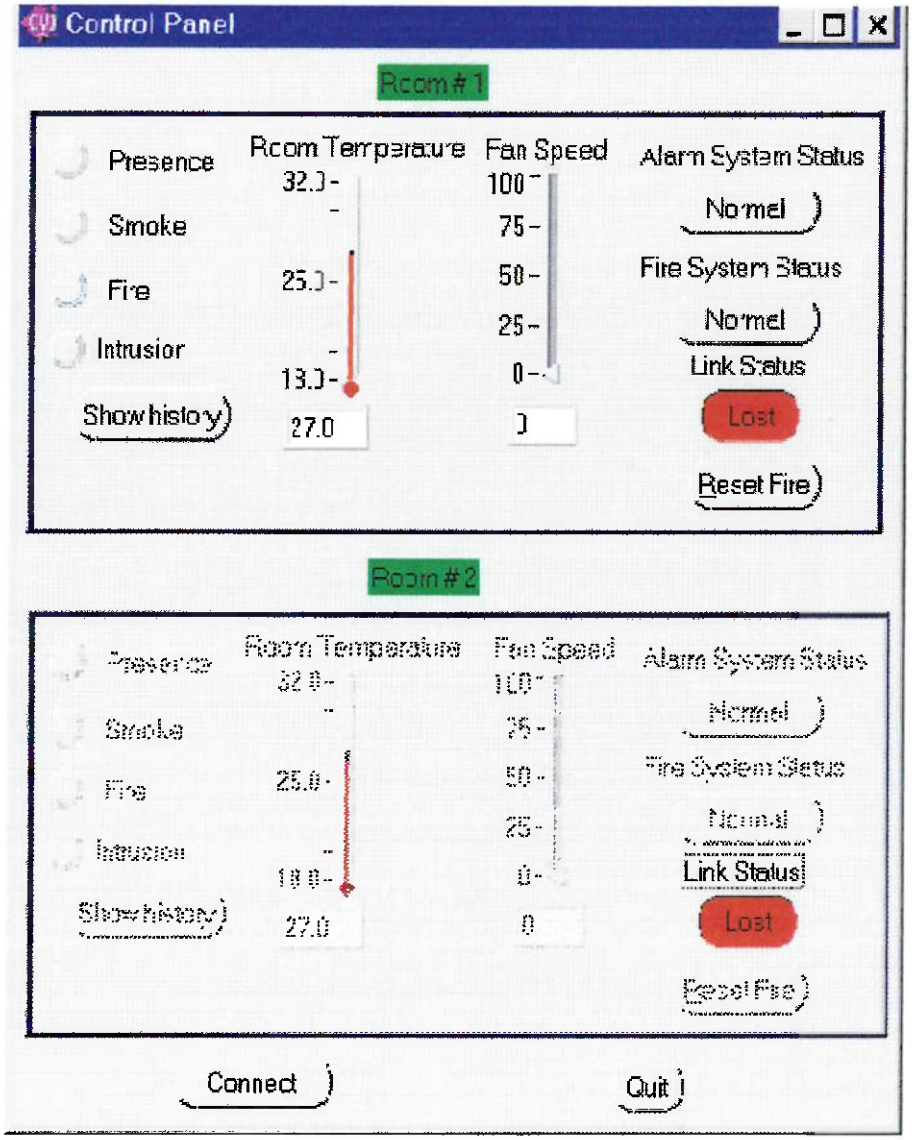


Figura A.2 - Tela do Control Panel

A tela *Control Status* permite que o usuário visualize o estado atual da sala, ou seja, o estado dos sensores e atuadores, e os sinais enviados pelo CLP ao nível supervisório. Além disso, possui controles que permitem que o usuário simule um nível superior de controle, enviando sinais de controle à planta.

Da mesma forma que na tela *User Input*, os mesmos controles e mostradores estão presentes para cada uma das salas. A seguir, encontra-se uma descrição mais detalhada de cada um desses componentes:

- Indicador *Intrusion*: Indica que há um intruso na sala.
- Indicador *Smoke*: Indica que há fumaça na sala.
- Indicador *Fire*: Indica que há indícios de incêndio na sala.
- Indicador *Presence*: Indica que a sala está ocupada.
- Indicador *Room Temperature*: Permite que se visualize a temperatura atual em cada uma das salas. Corresponde ao sinal do sensor de temperatura conectado ao CLP.
- Indicador *Fan Speed*: Esse indicador mostra a velocidade do ventilador em termos porcentuais com relação a velocidade máxima do ventilador.
- Botão *Show History*: Esse botão permite que se visualize o histórico de temperatura e velocidade do ventilador.
- Indicador *Link Status*: Esse botão indica qual é o estado da comunicação entre o CLP e a sistema supervisor.
- *Alarm System Status*: Esse botão permite que se alterne o estado da sala entre *Normal* e *Emergency*. No estado de emergência, os alarmes serão acionados.
- *Fire System Status*: Esse botão permite que se alterne o estado da sala entre *Normal* e *Fire*. No estado de incêndio, os sprinklers serão acionados.
- *Reset Fire*: Esse botão envia um sinal ao módulo de controle (CLP) indicando que a situação normal foi reestabelecida após uma indicação de suspeita de incêndio.

- Botão *Connect*: Esse botão permite que se estabeleça a conexão entre o CLP e o programa supervisor. Se necessário, o módulo *DDE Server* será automaticamente inicializado.
- Botão *Quit*: Esse botão encerra o programa supervisor. Caso o CLP continue em operação, passará para o modo *off-line*, operando sem comunicação com o supervisor.

Anexo A.3 Tela de Histórico de Temperatura e Velocidade:

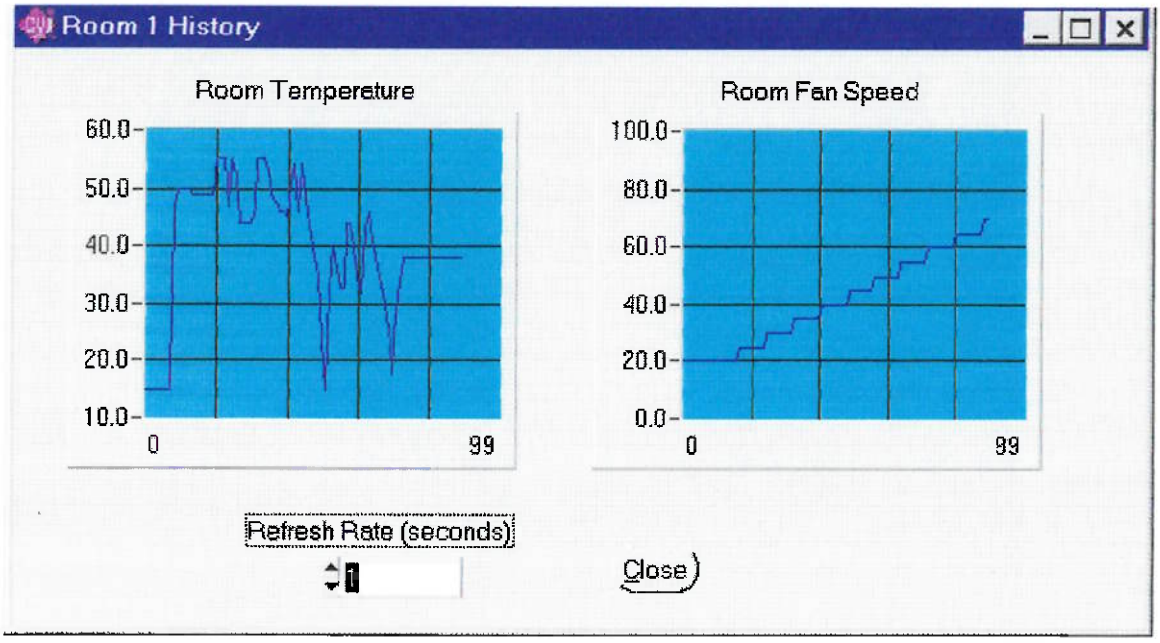


Figura A.3 - Tela do *Show History*

Nessa tela, pode-se visualizar o histórico de temperatura e velocidade do ventilador. Possui dois gráficos, onde são mostrados os históricos de temperatura e velocidade do ventilador, uma caixa de diálogo que permite se ajustar a freqüência com que os gráficos são atualizados e um botão para que se retorne às telas principais.

Bibliografia

[1] ARDITI, DAVID A. e NALBANTGIL, ECEM. **Perception of Trends in Intelligent Building Development**

[2] GOMES, LUIS, et al, **Campus-Guard: a domot targeted for integrated building monitoring and control.**

[3] McNAMARA, E. T. **Management Information Systems for Engineering Services Integration in Intelligent Buildings**

[4] RIES, A. MAHDAVI **How Intelligent is Our View of Intelligent Buildings?**

[5] SILVA, JOSÉ REINALDO, e RAMOS, ROBERTO L. C. BARROSO RAMOS. **Controle Integrado: Aplicação em Sistemas Prediais.** 3° SBAI , Vitória, 1997.

[6] SILVA, JOSÉ REINALDO, e MIYAGI, PAULO E. **Ambiente para Desenvolvimento e Teste de Técnicas de Integração de Sistemas em Edifícios Inteligentes.** 3° SBAI , Vitória, 1997.

[7] SILVA, JOSÉ REINALDO, MIYAGI, PAULO E. **PFS/MFG: A High Level Net for the Modeling of Discrete Manufacturing Systems .** in Balanced Automation Systems, IEEE/ECLA/IFIP Proceedings, vol. I - Architectures and desgin methods, 1995.

[8] SILVA, JOSÉ REINALDO, RAMOS, R.L.C.B., MIYAGI, PAULO E. - **Supervisory Control of integrated building systems: a balanced approach. in Balanced Automation Systems**, IEEE/ECLA/IFIP Proceedings, vol. II - Implementation Challenges for anthropocentric manufacturing, 1995.

Apêndice I. Ligações Elétricas

O presente trabalho de formatura tem como um dos objetivos construir uma maquete com sensores e atuadores de forma a simular um ambiente como aquele que seria encontrado em uma aplicação prática, ou seja, salas de um ambiente predial. A maquete em questão constitui-se de duas salas, cada qual com seus sensores, atuadores e um CLP.

Para que os sensores e atuadores funcionem da maneira esperada, deve-se elaborar o circuito de ligação, levando-se em conta as tensões e correntes mínimas e máximas para garantir o bom funcionamento dos componentes.

Deve-se levar em conta também os sinais de saída, verificando-se se é necessário acoplar um circuito de amplificação de potência ou de filtragem do sinal, qual a forma do sinal de saída (se é um pulso, qual o seu valor, se o sensor simplesmente atua como uma chave, etc.).

Apêndice I.1 Fontes de Alimentação

Para o sistema em questão, será necessário tanto ter disponível tensão alternada, quanto tensão contínua.

Como fonte de tensão alternada, utilizou-se a própria rede elétrica, amplamente disponível. A tensão elétrica de rede em São Paulo fornece uma tensão de 110V, com freqüência de 60Hz.

A fim de fornecer tensão contínua, dispôs-se de duas fontes de alimentação. Cada uma delas têm duas saídas: uma de 0 a 12V, e outra de 0 a –12V. Utilizou-se uma como uma fonte de 24V (ligando-se a referência do terra na parte negativa), e a outra como fonte de 12V.

Apêndice I.2 Componentes

Atuadores: Para simular o sistema de iluminação, cada sala da maquete contará com uma lâmpada elétrica incandescente, de 110V e 5W de potência. Estas lâmpadas devem ser ligadas à rede elétrica, e seu acionamento então deve ser feito através de um relé.

Um relé, como se sabe, nada mais é do que uma chave, acionada por uma bobina. Quando uma tensão suficiente é aplicada nos terminais da bobina, ela aciona a chave. Um relé pode ser normalmente aberto (conhecido como NA - quando não há tensão nos terminais da bobina se comporta como uma chave aberta, fechando quando aplica-se tensão nos terminais) ou normalmente fechado (ou NF, com comportamento inverso ao anterior).

As características mais importantes de um relé são a tensão e corrente de acionamento, a tensão que ele conduz e a freqüência com que pode ser acionado. Na aplicação em questão a freqüência não é relevante, já que o acionamento ocorrerá em freqüência bastante baixa.

Como o sinal de comando dos relés será dado pelo CLP, adotaram-se relés cuja tensão e corrente de acionamento fossem compatíveis com a saída digital do CLP.

Portanto, os relés adotados são da marca Schrack, tendo como tensão de acionamento 24V, e conduz tensão de 110V.

A montagem dos relés e das lâmpadas pode ser vista na figura I.1.

O alarme de emergência funciona com tensão de 12V. Como a saída do CLP fornece 10V, decidiu-se acioná-lo também com relés. Assim, o alarme está ligado à fonte

de 12VCC, tendo um relé como chave liga/desliga. O relé utilizado para o acionamento do alarme é igual ao acionamento das lâmpadas, já que o sinal de comando tem a mesma forma, e a tensão que ele deve conduzir é menor do que os 110V da lâmpada.

Os ventiladores utilizados também funcionam com corrente 12V CC. Mas para o seu acionamento será feito através de um PWM, como já discutido anteriormente. No escopo deste item vale apenas salientar que tanto o PWM quanto o motor serão alimentados com 12VCC. O sinal enviado ao PWM partirá da saída analógica do CLP, e o PWM enviará um sinal de tensão correspondente ao ventilador, permitindo o controle de sua velocidade.

Apêndice I.3 Considerações sobre potência e filtragem

Pode-se observar que os sinal obtidos pelos próprios componentes do circuito eram compatíveis entre si, não havendo a necessidade de circuitos de amplificação de potência ou sinal nem de filtragem.

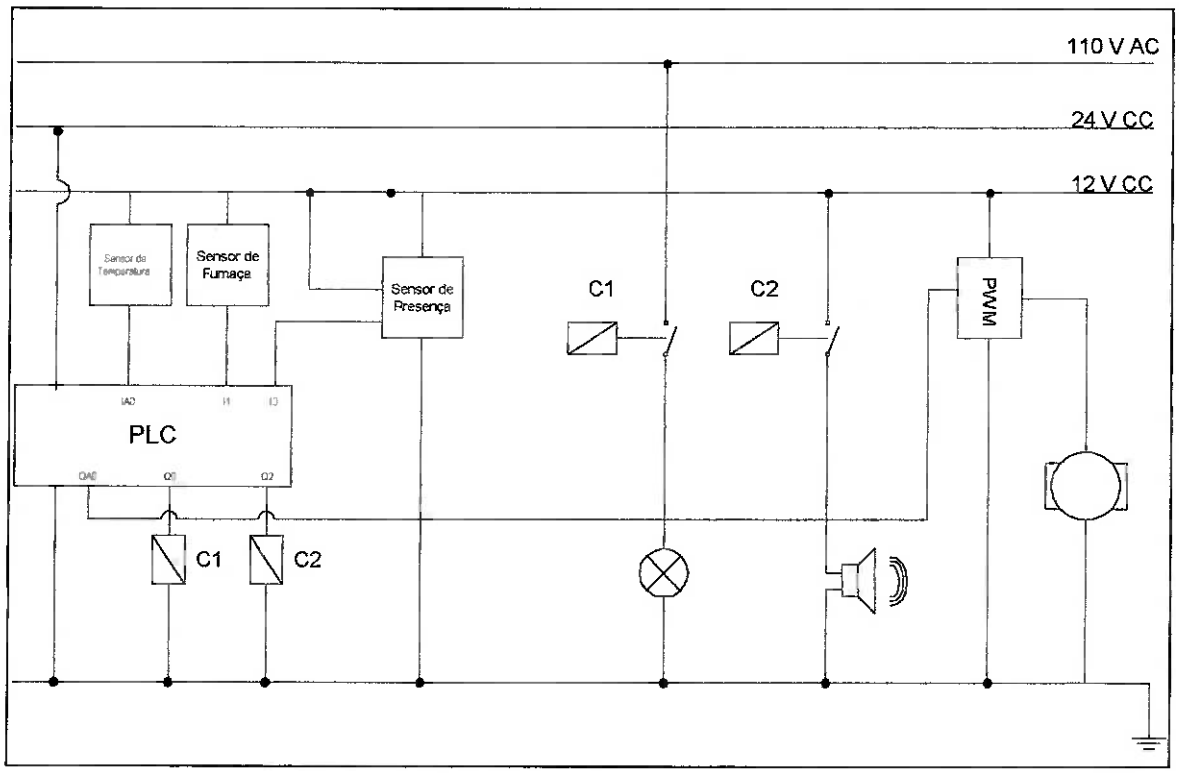


Figura I.1 – Esquema elétrico de uma das salas da maquete

Apêndice II. PWM

O método normalmente utilizado para controle do acionamento de motores elétricos é utilização de um circuito PWM - *Pulse Width Modulator*, que faz uma modulação de voltagem em através da variação da largura de pulso de voltagem.

O sinal de entrada varia entre 0 e 100% de uma voltagem de controle, enquanto que o sinal de saída está em 0 ou 100% da voltagem máxima. O que varia é o intervalo de tempo em que o sinal de saída fica em 0 ou em 100%. Isso é obtido através da comparação do sinal de entrada com a onda de forma triangular. Conforme o sinal de entrada aumente ou diminua o intervalo em que a onda de saída se mantém em 100% aumenta ou diminui, respectivamente.

No sistema desenvolvido, utilizou-se um circuito integrado, amplamente disponível no mercado: o 3524. Este CI já possui um oscilador responsável pela geração do sinal de onda triangular, e permite a entrada de um sinal de voltagem invertido ou não invertido.

Abaixo vemos o esquema do circuito implementado:

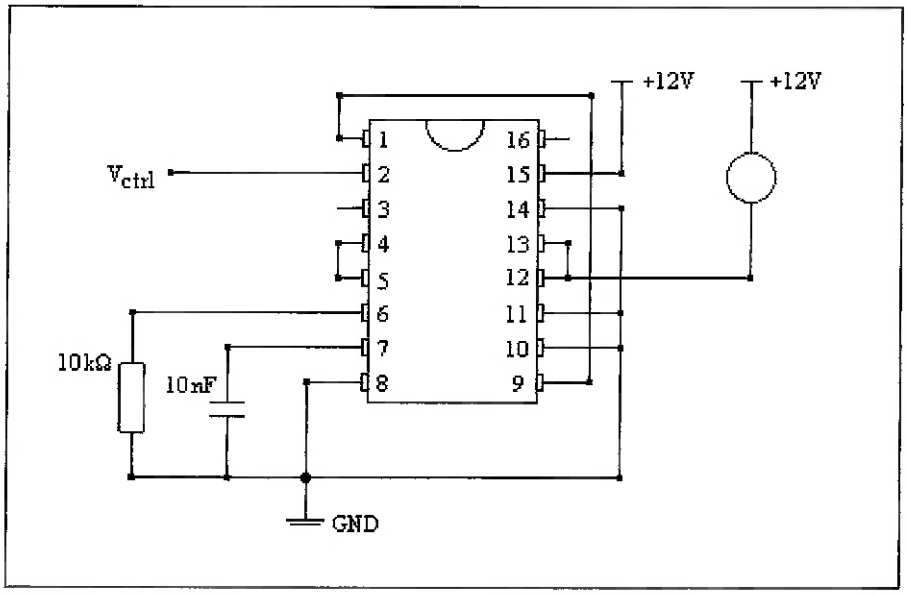


Figura II.1 - Esquema do PWM

Apêndice III. Descrição das funções do mód. de controle (CLP):

O módulo de controle presente no CLP segue o funcionamento descrito pelo diagrama abaixo:

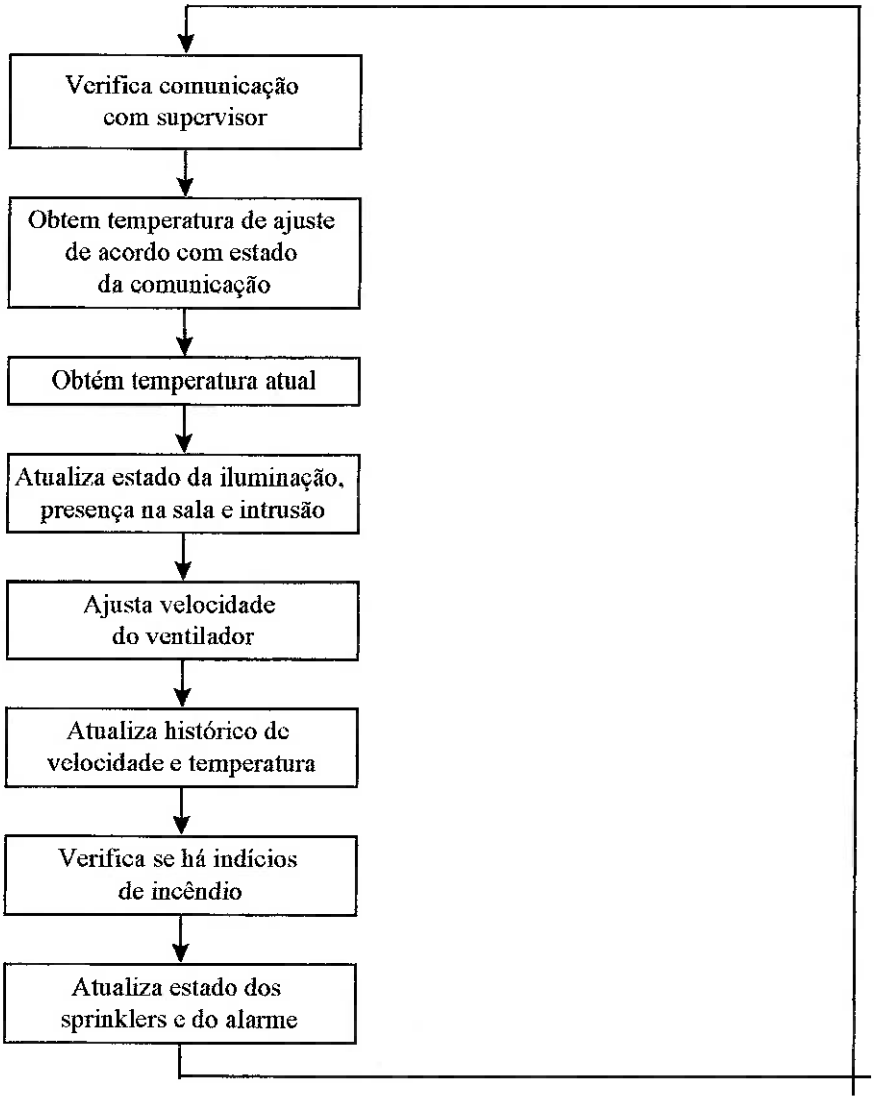


Figura III.1 - Fluxograma do programa no CLP

Nota-se que para realizar as funções descritas no diagrama acima, foram desenvolvidas algumas funções (*Function Blocks*) que são descritos a seguir.

Apêndice III.1 **Função FCNCOMUN:**

Essa função recebe como sinal o estado atual do sinal de verificação de comunicação enviado pelo supervisor e retorna o estado atual da comunicação. Internamente, possui uma variável que indica se houve alteração nesse sinal desde a última atualização e um temporizador para gerar o período em que se deve atualizar o estado da variável de estado de comunicação. Para se entender melhor o funcionamento dessa função é conveniente se verificar a descrição do algoritmo de verificação de comunicaçào descrito em 7.3

Abaixo, encontra-se o diagrama de funcionamento desse bloco:

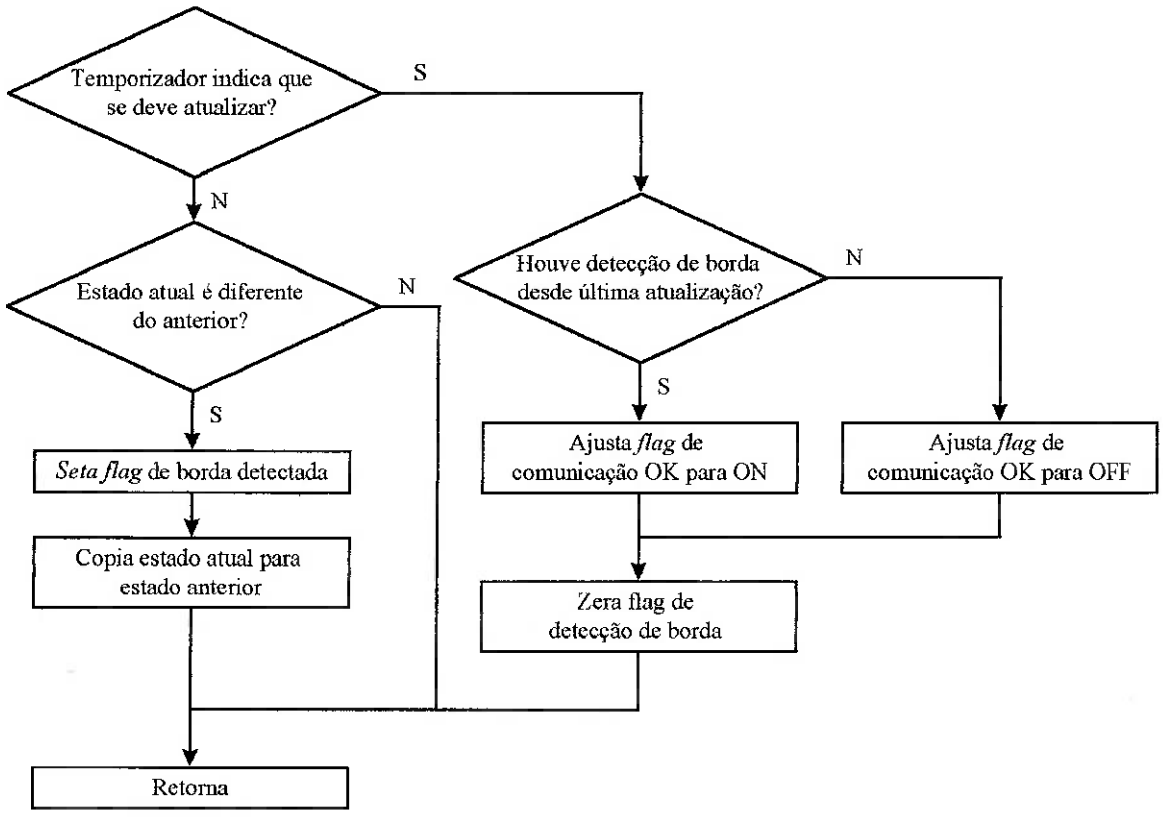


Figura III.2 - Fluxograma da função FCNCOMUN

Apêndice III.2 Função FCNLDTEMP:

Essa função permite que se faça a atualização das temperaturas máxima e mínima adminssíveis na sala. Para isso, recebe como parâmetros a temperatura de ajuste enviada pelo supervisor, a temperatura ajustada no potenciômentro do próprio CLP e o estado da comunicação. A partir desses dados, a função decide se o ajuste a ser levado em conta é o local (potenciômetro) ou o enviado pelo supervisor.

Abaixo segue o diagrama que descreve o processamento feito pela função:

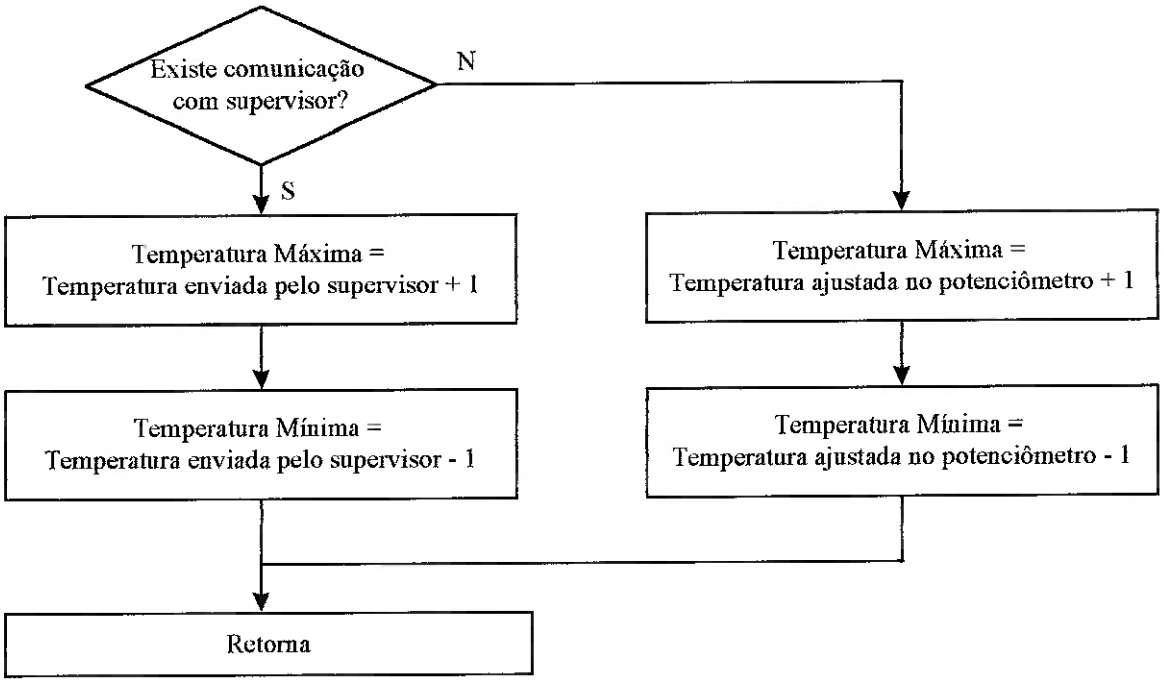


Figura III.3 - Fluxograma da função FCNLDTEMP

Apêndice III.3 Função FCNINTRU:

Essa função recebe como parâmetros a indicação de se há presença na sala e o estado da sala (habilitada/desabilitada). A partir desses sinais, fornece qual deve ser o

estado se a sala está ocupada, se há um intruso na sala e se as luzes devem ser acionadas ou não. Para isso, segue o diagrama mostrado abaixo:

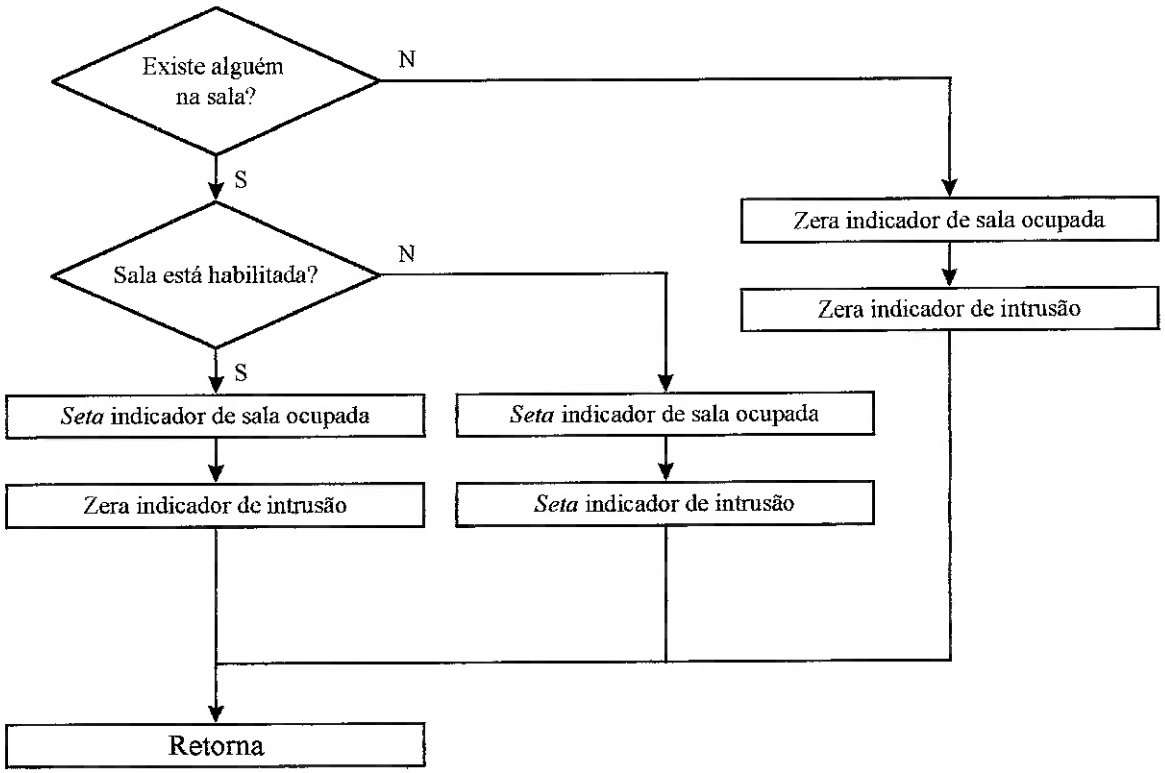


Figura III.4 - Fluxograma da função FCNINTRU

Apêndice III.4 Função FCNALARM;

Para essa função devem ser fornecidos os seguintes sinais:

- Indicador de possível incêndio na sala;
- Estado da comunicação entre CLP e supervisor;
- Sinal de acionamento de estado de emergência do supervisor;
- Sinal de acionamento dos sprinklers na sala enviado pelo supervisor;
- Sinal de retorno ao estado normal enviado pelo supervisor;

- Sinal de desligamento dos sprinklers acionado localmente;

A partir desses sinais a função gera o estado de acionamento dos alarmes e dos sprinklers. Para isso, decide se os sinais locais ou os enviados pelo supervisor devem ser levados em conta. Isso pode ser visto no seu diagrama de funcionamento mostrado a seguir:

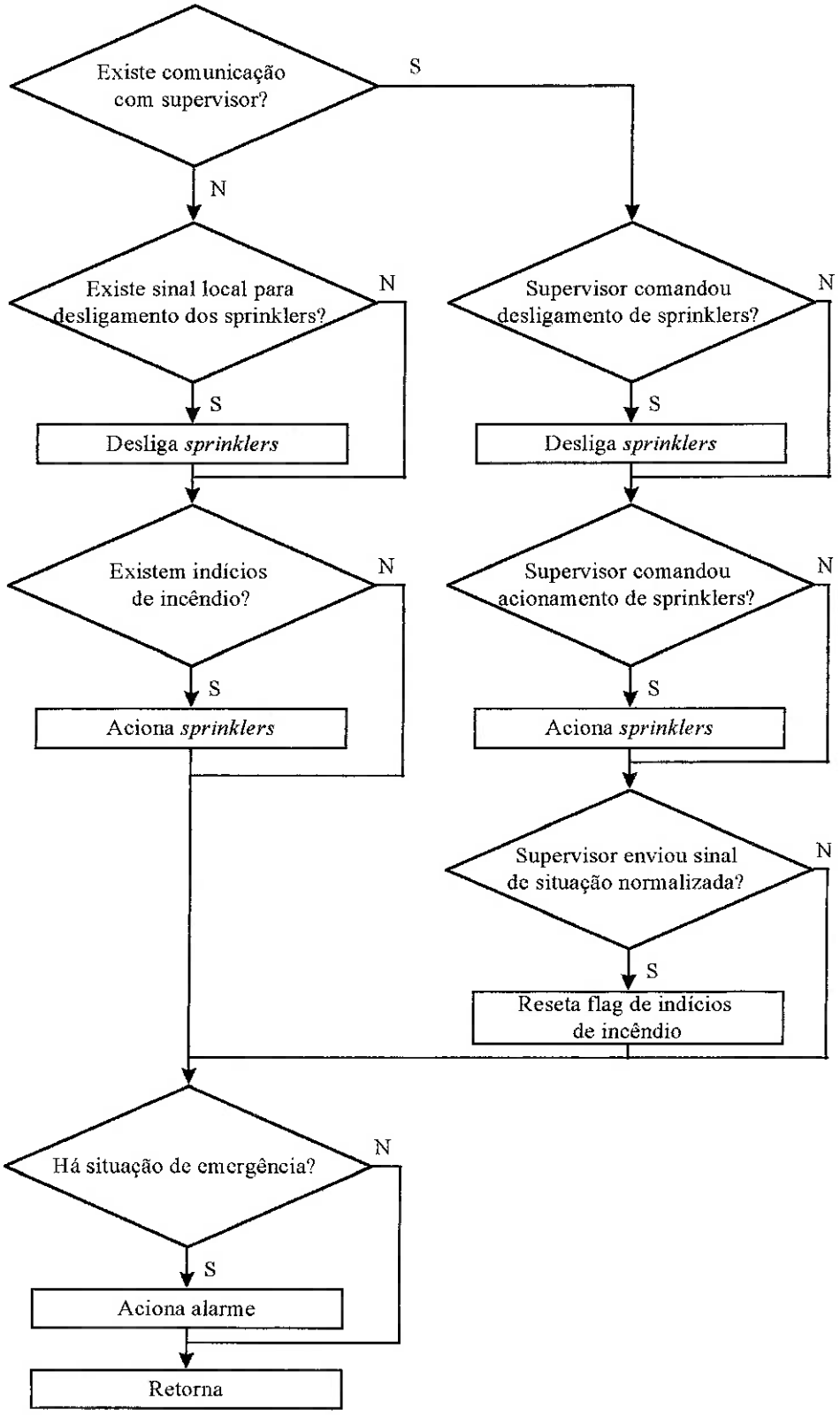


Figura III.5 - Fluxograma da função FCNALARM

Apêndice III.5 Função FCNVENTIL

Essa função realiza o ajuste da velocidade do ventilador de acordo com as condições do ambiente. Para isso, recebe as seguintes variáveis:

- Temperatura atual da sala
- Máxima temperatura admissível na sala
- Mínima temperatura admissível na sala
- Presença na sala
- Sala habilitada
- Velocidade atual do ventilador
- Incremento da velocidade do ventilador
- Velocidade fixa de operação

Internamente, essa função possui um temporizador que faz com que a cada vez que a velocidade seja alterada, aguarde-se um intervalo de 30s até que possa ser novamente ajustada. Isso permite que a velocidade do ventilador não passe quase instantaneamente para 0 ou 100% conforme o caso.

O retorno dessa função é o novo valor percentual da velocidade do ventilador. A seguir, mostra-se o fluxograma responsável pela execução da função:

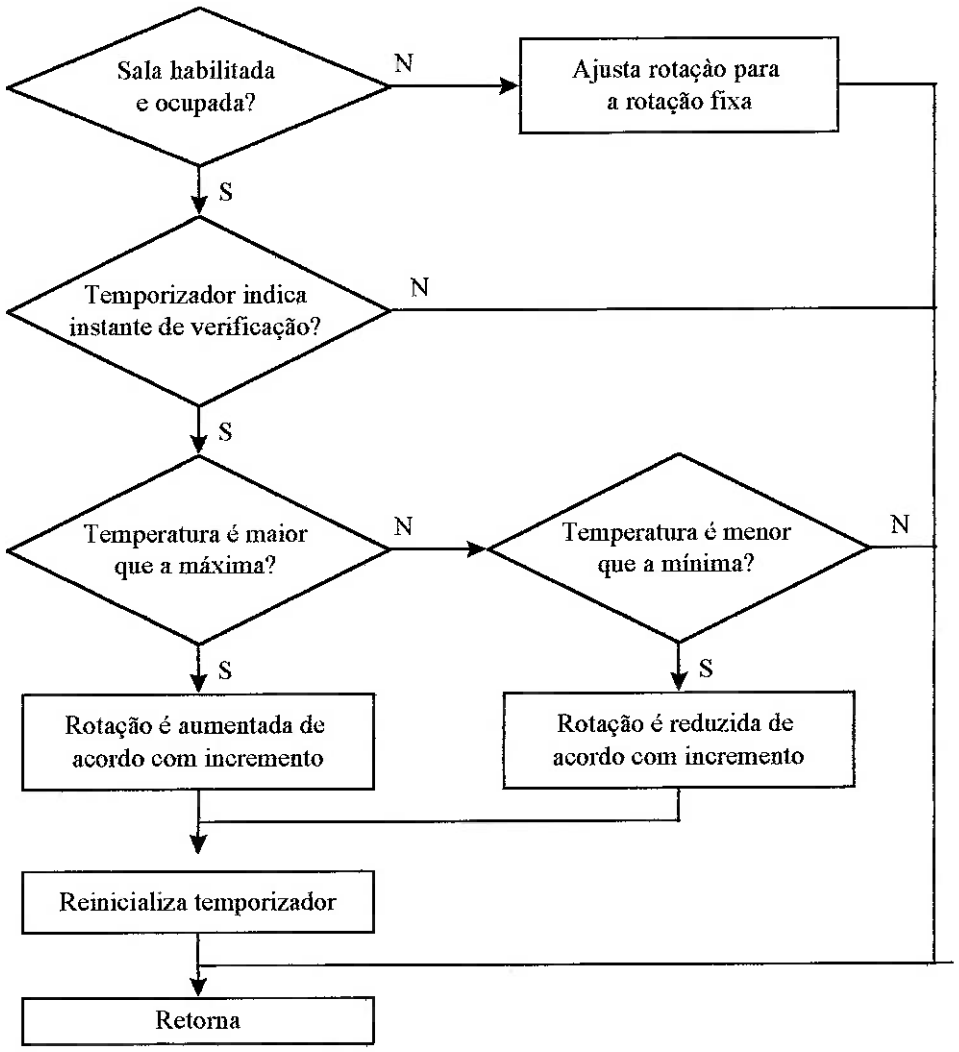


Figura III.6 - Fluxograma da função FCNVENTIL

Apêndice III.6 Função FCNARRAY:

Essa função recebe como parâmetros a velocidade atual do ventilador e a temperatura atual na sala. Internamente, possui um temporizador que fará com que esses valores sejam adicionados aos vetores de histórico de temperatura e velocidade a cada 30s. Cada um desses vetores possui 20 elementos correspondendo, portanto, ao histórico

dos últimos 10 minutos. A partir dos dados nesses vetores, a função indica se cada um dos históricos (de temperatura e de velocidade) são crescentes nesse intervalo de tempo.

O diagrama para executar essas ações é mostrado abaixo:

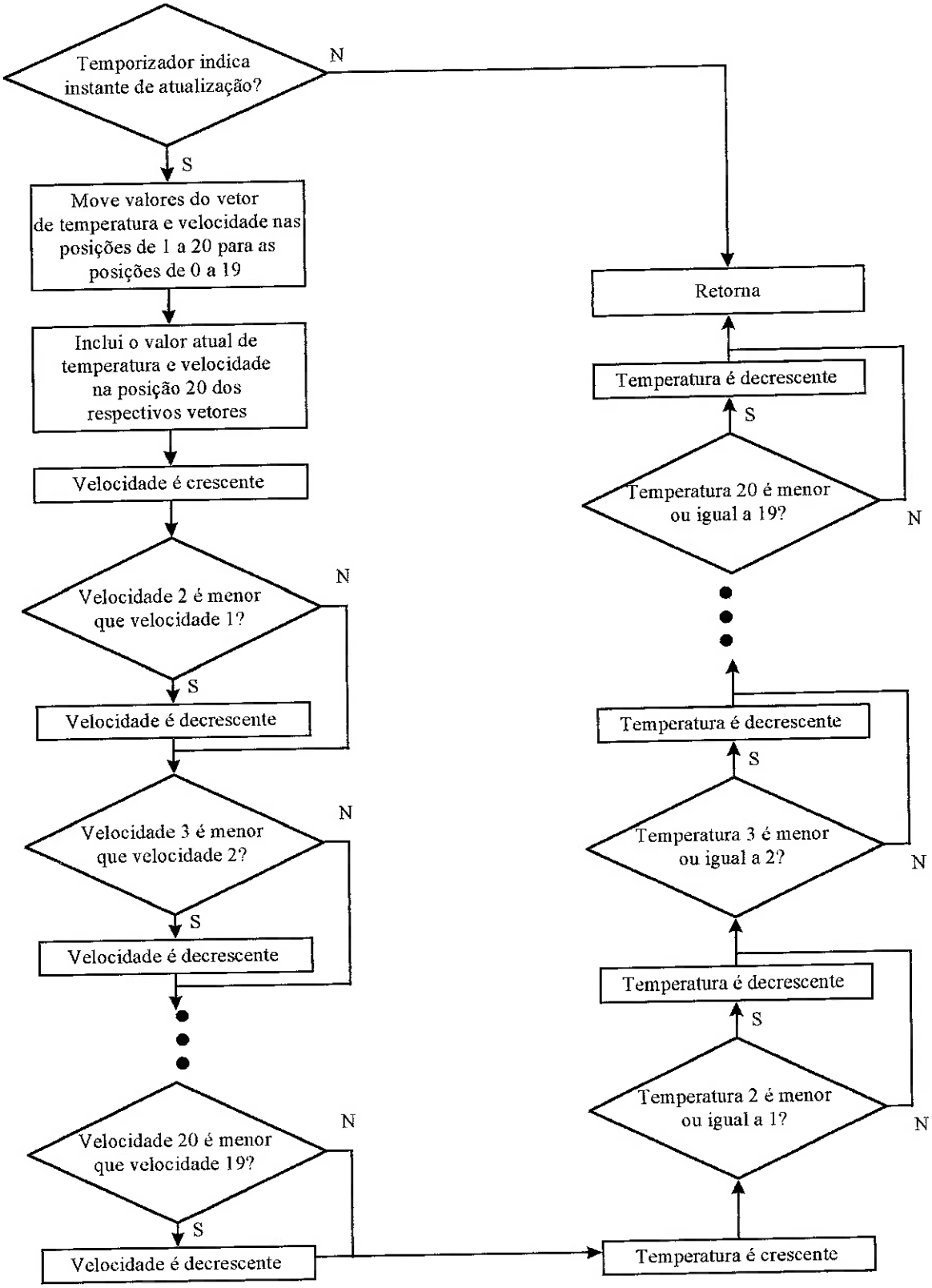


Figura III.7 - Fluxograma da função FCNARRAY

Apêndice III.7 Função SPDCONVERT e TEMPCONVERT:

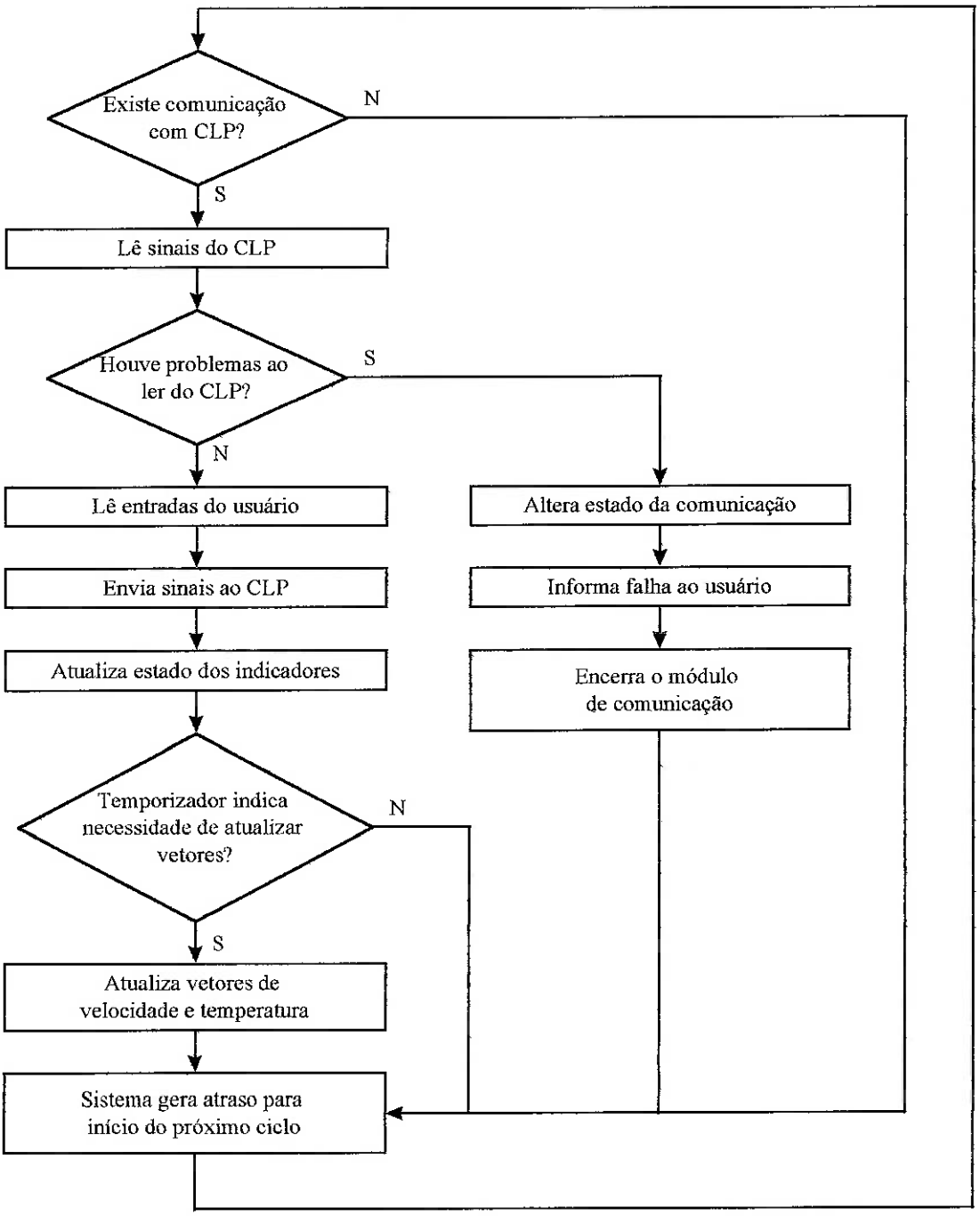
A função SPDCONVERT realiza a conversão de um número que indica a velocidade porcentual do ventilador em um número de 10 bits correspondente à saída analógica do CLP.

A função TEMPCONVERT recebe os limites de um *range* de temperatura em °C e um número de 11 bits correspondente à entrada analógica do CLP. Como retorno ela fornece a temperatura em °C correspondente a essa entrada admitindo que o limite superior do range de temperatura seja correspondente a 10V e o inferior a 0V

Apêndice IV. Fluxograma do ciclo de controle do módulo supervisor:

Como já descrito em 7.1, o programa supervisor se encarrega de fazer a interface entre o usuário e o controlador (CLP). Para isso realiza ciclos periódicos onde faz a leitura do estado do controlador e dos comandos gerados pelo usuário e, a partir daí, atualiza os dados mostrados e envia sinais de comando ao CLP.

A seguir mostra-se o fluxograma que descreve o funcionamento desse ciclo. Note-se que, em caso de falhas na comunicação, o supervisor toma atitudes cabíveis e informa o usuário sobre essa falha.



Obs.: O estado da comunicação pode ser alterado para OK fora desse ciclo, através do botão *Connect*.

Figura IV.1 - Fluxograma do ciclo do programa Supervisor

Apêndice V. Listagem das funções do módulo de controle (CLP):

A seguir, Sào mostradas as listagens do programa principal e das funções auxiliares do software desenvolvido para realizar o controle da planta:

Apêndice V.1 Programa Principal

```
PROGRAM BLOCK009
VAR

        (* Verificacao de comunicacao com supervisorio *)
        ComunicOK : BOOL := 0;
        UltSinal  : BYTE := 0;
        SinalOK   : BOOL := 0;

        (* Variaveis Enviadas ao Supervisorio *)
        Incendio      AT %M0.0.0.20.0 : BOOL:=0;
        Intrusao      AT %M0.0.0.22.0 : BOOL;
        SalaOcupada   AT %M0.0.0.24.0 : BOOL;
        TempAtual     AT %MB0.0.0.26  : BYTE;
        VelAtual      AT %MB0.0.0.28  : BYTE;
        FumacaSala    AT %MB0.0.0.30  : BYTE;
        AckSitNormal AT %M0.0.0.32.0  : BOOL:=0;

        (* Entradas provenientes do supervisorio *)
        SalaHabilitada AT %M0.0.0.4.0  : BOOL;
        Emergencia     AT %M0.0.0.6.0  : BOOL;
        UserTemp       AT %MB0.0.0.8   : BYTE;
        SinalAtual     AT %MB0.0.0.10  : BYTE;
        SituacacNormal AT %M0.0.0.12.0 : BOOL;
        SupervIncendio AT %M0.0.0.14.0 : BOOL;
        IncrementoPerc AT %MB0.0.0.16  : BYTE := 5;
        VelocFixa      AT %MB0.0.0.18  : BYTE := 20;

        MaxTemp : UINT;
        MinTemp : UINT;

        (* Entradas da planta *)

        Presenca          AT %I0.0.0.0.0 : BOOL;
        Fumaca            AT %I0.0.0.0.1 : BOOL;
        Desliga_Sprinkler AT %I0.0.0.0.7 : BOOL;
        Reseta_Incendio          AT %I0.0.0.0.6 : BOOL;
        IA00     AT %IAW0.0.0.0 : WORD;
        IA04     AT %IAW0.0.0.4 : WORD;

        TempPotenc  : UINT;
        Temperatura : UINT;

        (* Historico de temperatura e velocidade *)
        Array_Temp: ARRAY[1..20] OF UINT;
        Array_Vel:  ARRAY[1..20] OF INT;

        TempCrescente : BOOL;
        VelCrescente  : BOOL;

        (* Saidas enviadas para a planta *)
        Luzes      AT %Q0.0.0.0.0 : BOOL;
        Alarme     AT %Q0.0.0.0.1 : BOOL;
        Sprinkler AT %Q0.0.0.0.2 : BOOL;
        VelocVent  AT %QAW0.0.0.0 : WORD;

        Q06 AT %Q0.0.0.0.6 : BOOL;
        Q07 AT %Q0.0.0.0.7 : BOOL;
```

```
        (* Parametros *)
        TEMP_RANGEMAX: UINT := 55;
        TEMP_RANGEMIN: UINT := 15;
        Incremento   : INT;

        (* Variaveis auxiliares *)
        VelocPerc    : INT  ;
        WDVelocPerc : WORD ;
        TempEMax     : BOOL := 0;

        (* Funcoes e variaveis de temporizacao *)
        TimerGetTemp : TP;
        DTempGetTemp : TIME := T#5s;
        TrigGetTemp  : BOOL := 0;

        (* Funcoes de Controle *)
        Ctrl_Alarme : FCNALARM;
        Ctrl_Acesso : FCNINTRU;
        Ctrl_Temp   : FCNVENTIL;
        Ctrl_Array  : FCNARRAY;

        VerificaComunic : FCNCOMUN;
        LoadInputTemp   : FCNLDTMP;

        (* Funcoes de conversao *)
        SPEEDCONVERT    : SPDCONV;
        CURRENTTEMP     : TEMPCONV;

END_VAR
(* Verifica Comunicacao *)
CAL     VerificaComunic(In_SinalAtual := SinalAtual)
LD      VerificaComunic.Out_ComunicOK
ST      ComunicOK

(* Entradas provenientes do supervisorio *)
CarregaUserTemp:
CAL     LoadInputTemp(In_SupervTemp := UserTemp,
                      In_PotencTemp := IA00,
                      In_ComunicOK := ComunicOK)
LD      LoadInputTemp.Out_MaxTemp
ST      MaxTemp
LD      LoadInputTemp.Out_MinTemp
ST      MinTemp

LoadTemperatura:
CAL     CURRENTTEMP(TEMPWORD:= IA04,
                    TEMPMAX := TEMP_RANGEMAX,
                    TEMPMIN := TEMP_RANGEMIN)
LD      CURRENTTEMP.TEMPGRAUS
ST      TempAtual
BYTE_TO_UINT
ST      Temperatura                    (* Temperatura na Sala em oC *)

(* Controle de acesso e iluminacao *)
CAL     Ctrl_Acesso(In_Presenca := Presenca,
                    In_SalaHabilitada := SalaHabilitada)
LD      Ctrl_Acesso.Out_SalaOcupada
ST      SalaOcupada
LD      Ctrl_Acesso.Out_Intrusao
ST      Intrusao
LD      Ctrl_Acesso.Out_Luzes
ST      Luzes


(* Controle do ventilador *)
CAL     Ctrl_Temp(In_Presenca := Presenca,
                  In_SalaHabilitada := SalaHabilitada,
                  In_MaxTemp := MaxTemp,
                  In_MinTemp := MinTemp,
                  In_Temperatura := Temperatura,
                  In_IncrementoPerc := IncrementoPerc,
                  In_VelocFixa := VelocFixa,
                  Ext_VelocPerc := VelocPerc)

LD      VelocPerc
```

```
        INT_TO_BYTE
        ST      VelAtual
        LD      VelocPerc
        INT_TO_WORD
        ST      WDVelocPerc
        CAL     SPEEDCONVERT(SPDPC := WDVelocPerc)
        LD      SPEEDCONVERT.SPDBITS
        ST      VelocVent

        AtualizaArrays:
        CAL     Ctrl_Array (In_Temperatura := Temperatura,
                            In_VelocPerc := VelocPerc)
        LD      Ctrl_Array.Out_TempCrescente
        ST      TempCrescente
        LD      Ctrl_Array.Out_VelCrescente
        ST      VelCrescente

        LD      0
        ST      TempEMax
        LD      Temperatura
        FQ      TEMP_RANGEMAX
        S       TempEMax

        (* Verifica Incendio *)
        LD      Fumaca
        JMPCN   FimVerificaIncendio
        LD      TempCrescente
        AND     VelCrescente
        OR      TempEMax
        JMPCN   FimVerificaIncendio
        LD      1
        ST      Incendio

        FimVerificaIncendio:

        CAL     Ctrl_Alarme(Ext_Incendio:= Incendio,
                            In_SupervIncendio:= SupervIncendio,
                            In_ComunicOK := ComunicOK,
                            In_SituacaoNormal := SituacaoNormal,
                            In_Emergencia := Emergencia,
                            In_Desliga_Sprinkler := Desliga_Sprinkler,
                            In_Reseta_Incendio := Reseta_Incendio,
                            Ext_AckSitNormal := AckSitNormal)
        LD      Ctrl_Alarme.Out_Alarme
        ST      Alarme
        LD      Ctrl_Alarme.Out_Sprinkler
        ST      Sprinkler

        (* Obs: O CLP so tome atitudes autonomas no sentido de acionar os sprinklers
                *)
        (*      O seu desligamento so sera feito pelo supervisor ou <manualmente>
        *)
        (*      Com isso, se garante que um erro de avaliacao no nivel do CLP nao
prejudique a segurana        *)

        LD      Fumaca
        ST      FumacaSala.0

        (* Permite visualizacao do estado da comunicacao *)
        SendComunicOK:
        LD      ComunicOK
        ST      Q07
        END_PROGRAM
```

Apêndice V.2 Função FCNALARM;

```
FUNCTION_BLOCK FCNALARM
VAR_INPUT
        In_ComunicOK : BOOL ;
        In_SituacaoNormal : BOOL ;
        In_Desliga_Sprinkler : BOOL ;
        In_Reseta_Incendio : BOOL;
        In_Emergencia : BOOL ;
        In_SupervIncendio : BOOL ;
END_VAR
VAR_OUTPUT
        Out_Alarme : BOOL ;
        Out_Sprinkler : BOOL ;
        Out_Incendio  : BOOL ;
        Out_SituacaoNormal: BOOL;
END_VAR
VAR_IN_OUT
        Ext_AckSitNormal : BOOL;
        Ext_Incendio   : BOOL;
END_VAR
VAR
        OldSituacaoNormal : BOOL;
END_VAR
LD      Ext_AckSitNormal
ANDN    In_SituacaoNormal
R       Ext_AckSitNormal

(* Comunicacao OK *)

LD      In_ComunicOK
JMPCN   ComunicLost

LD      In_SituacaoNormal
ST      Ext_AckSitNormal

LD      In_SituacaoNormal
ANDN    In_SupervIncendio
R       Out_Sprinkler
R       Ext_Incendio

LD      In_SupervIncendio
S       Out_Sprinkler


JMP     Fim_Incendio

(* Comunicacao Perdida *)

ComunicLost:

LD      Ext_Incendio
S       Out_Sprinkler

LD      In_Desliga_Sprinkler
R       Out_Sprinkler

LD      In_Reseta_Incendio
R       Ext_Incendio

Fim_Incendio:

LD      In_Emergencia
S       Out_Alarme

LDN     In_Emergencia
R       Out_Alarme

LD      In_SituacaoNormal
ST      Out_SituacaoNormal
```

```
LD      Ext_AckSitNormal
R       Out_SituacaoNormal

END_FUNCTION_BLOCK
```

Apêndice V.3 Função FCNARRAY;

```
FUNCTION_BLOCK FCNARRAY
VAR_INPUT
        In_Temperatura : UINT ;
        In_VelocPerc : INT;
END_VAR
VAR_OUTPUT
        Out_TempCrescente : BOOL;
        Out_VelCrescente : BOOL;
END_VAR
VAR
        Array_Temp : ARRAY [1..20 ] OF UINT ;
        Array_Vel : ARRAY [1..20] OF INT;
        TrigGetTemp : BOOL := 0 ;
        DTempGetTemp : TIME := T#1s ;
        TimerGetTemp : TP ;
END_VAR
AtualizaArrays:
CAL     TimerGetTemp(IN := TrigGetTemp,
                     PT := DTempGetTemp)

TTemp1:
LD      TimerGetTemp.Q
JMPC    TTemp0
LD      1
ST      TrigGetTemp

LD      Array_Temp[2]
ST      Array_Temp[1]

LD      Array_Temp[3]
ST      Array_Temp[2]

LD      Array_Temp[4]
ST      Array_Temp[3]

LD      Array_Temp[5]
ST      Array_Temp[4]

LD      Array_Temp[6]
ST      Array_Temp[5]

LD      Array_Temp[7]
ST      Array_Temp[6]

LD      Array_Temp[8]
ST      Array_Temp[7]

LD      Array_Temp[9]
ST      Array_Temp[8]

LD      Array_Temp[10]
ST      Array_Temp[9]

LD      Array_Temp[11]
ST      Array_Temp[10]

LD      Array_Temp[12]
ST      Array_Temp[11]

LD      Array_Temp[13]
ST      Array_Temp[12]

LD      Array_Temp[14]
ST      Array_Temp[13]
```

```
LD      Array_Temp[15]
ST      Array_Temp[14]

LD      Array_Temp[16]
ST      Array_Temp[15]

LD      Array_Temp[17]
ST      Array_Temp[16]

LD      Array_Temp[18]
ST      Array_Temp[17]

LD      Array_Temp[19]
ST      Array_Temp[18]

LD      Array_Temp[20]
ST      Array_Temp[19]

LD      In_Temperatura
ST      Array_Temp[20]

LD      1
ST      Out_TempCrescente

Comp1:
LD      Array_Temp[2]
GT      Array_Temp[1]

JMPC    Comp2
LD      0
ST      Out_TempCrescente

Comp2:
LD      Array_Temp[3]
GT      Array_Temp[2]
JMPC    Comp3
LD      0
ST      Out_TempCrescente

Comp3:
LD      Array_Temp[4]
GT      Array_Temp[3]
JMPC    Comp4
LD      0
ST      Out_TempCrescente

Comp4:
LD      Array_Temp[5]
GT      Array_Temp[4]
JMPC    Comp5
LD      0
ST      Out_TempCrescente

Comp5:
LD      Array_Temp[6]
GT      Array_Temp[5]
JMPC    Comp6
LD      0
ST      Out_TempCrescente

Comp6:
LD      Array_Temp[7]
GT      Array_Temp[6]
JMPC    Comp7
LD      0
ST      Out_TempCrescente

Comp7:
LD      Array_Temp[8]
GT      Array_Temp[7]
JMPC    Comp8
LD      0
ST      Out_TempCrescente
```

```
Comp8:
LD      Array_Temp[9]
GT      Array_Temp[8]
JMPC    Comp9
LD      0
ST      Out_TempCrescente

Comp9:
LD      Array_Temp[10]
GT      Array_Temp[9]
JMPC    Comp10
LD      0
ST      Out_TempCrescente

Comp10:
LD      Array_Temp[11]
GT      Array_Temp[10]
JMPC    Comp11
LD      0
ST      Out_TempCrescente

Comp11:
LD      Array_Temp[12]
GT      Array_Temp[11]
JMPC    Comp12
LD      0
ST      Out_TempCrescente

Comp12:
LD      Array_Temp[13]
GT      Array_Temp[12]
JMPC    Comp13
LD      0
ST      Out_TempCrescente

Comp13:
LD      Array_Temp[14]
GT      Array_Temp[13]
JMPC    Comp14
LD      0
ST      Out_TempCrescente

Comp14:
LD      Array_Temp[15]
GT      Array_Temp[14]
JMPC    Comp15
LD      0
ST      Out_TempCrescente

Comp15:
LD      Array_Temp[16]
GT      Array_Temp[15]
JMPC    Comp16
LD      0
ST      Out_TempCrescente

Comp16:
LD      Array_Temp[17]
GT      Array_Temp[16]
JMPC    Comp17
LD      0
ST      Out_TempCrescente

Comp17:
LD      Array_Temp[18]
GT      Array_Temp[17]
JMPC    Comp18
LD      0
ST      Out_TempCrescente

Comp18:
LD      Array_Temp[19]
GT      Array_Temp[18]
JMPC    Comp19
LD      0
```

```
ST      Out_TempCrescente

Comp19:
LD      Array_Temp[20]
GT      Array_Temp[19]
JMPC    EndUpdateTemp
LD      0
ST      Out_TempCrescente
EndUpdateTemp:

LD      Array_Vel[2]
ST      Array_Vel[1]

LD      Array_Vel[3]
ST      Array_Vel[2]

LD      Array_Vel[4]
ST      Array_Vel[3]

LD      Array_Vel[5]
ST      Array_Vel[4]

LD      Array_Vel[6]
ST      Array_Vel[5]

LD      Array_Vel[7]
ST      Array_Vel[6]

LD      Array_Vel[8]
ST      Array_Vel[7]

LD      Array_Vel[9]
ST      Array_Vel[8]

LD      Array_Vel[10]
ST      Array_Vel[9]

LD      Array_Vel[11]
ST      Array_Vel[10]

LD      Array_Vel[12]
ST      Array_Vel[11]

LD      Array_Vel[13]
ST      Array_Vel[12]

LD      Array_Vel[14]
ST      Array_Vel[13]

LD      Array_Vel[15]
ST      Array_Vel[14]

LD      Array_Vel[16]
ST      Array_Vel[15]

LD      Array_Vel[17]
ST      Array_Vel[16]

LD      Array_Vel[18]
ST      Array_Vel[17]

LD      Array_Vel[19]
ST      Array_Vel[18]

LD      Array_Vel[20]
ST      Array_Vel[19]

LD      In_VelocPerc
ST      Array_Vel[20]

LD      1
ST      Out_VelCrescente

CompVel1:
LD      Array_Vel[2]
```

```
GE      Array_Vel[1]
JMPC    CompVel2
LD      0
ST      Out_VelCrescente

CompVel2:
LD      Array_Vel[3]
GE      Array_Vel[2]
JMPC    CompVel3
LD      0
ST      Out_VelCrescente

CompVel3:
LD      Array_Vel[4]
GE      Array_Vel[3]
JMPC    CompVel4
LD      0
ST      Out_VelCrescente

CompVel4:
LD      Array_Vel[5]
GE      Array_Vel[4]
JMPC    CompVel5
LD      0
ST      Out_VelCrescente

CompVel5:
LD      Array_Vel[6]
GE      Array_Vel[5]
JMPC    CompVel6
LD      0
ST      Out_VelCrescente

CompVel6:
LD      Array_Vel[7]
GE      Array_Vel[6]
JMPC    CompVel7
LD      0
ST      Out_VelCrescente

CompVel7:
LD      Array_Vel[8]
GE      Array_Vel[7]
JMPC    CompVel8
LD      0
ST      Out_VelCrescente

CompVel8:
LD      Array_Vel[9]
GE      Array_Vel[8]
JMPC    CompVel9
LD      0
ST      Out_VelCrescente

CompVel9:
LD      Array_Vel[10]
GE      Array_Vel[9]
JMPC    CompVel10
LD      0
ST      Out_VelCrescente

CompVel10:
LD      Array_Vel[11]
GE      Array_Vel[10]
JMPC    CompVel11
LD      0
ST      Out_VelCrescente

CompVel11:
LD      Array_Vel[12]
GE      Array_Vel[11]
JMPC    CompVel12
LD      0
ST      Out_VelCrescente
```

```
CompVel12:
LD      Array_Vel[13]
GE      Array_Vel[12]
JMPC    CompVel13
LD      0
ST      Out_VelCrescente

CompVel13:
LD      Array_Vel[14]
GE      Array_Vel[13]
JMPC    CompVel14
LD      0
ST      Out_VelCrescente

CompVel14:
LD      Array_Vel[15]
GE      Array_Vel[14]
JMPC    CompVel15
LD      0
ST      Out_VelCrescente

CompVel15:
LD      Array_Vel[16]
GE      Array_Vel[15]
JMPC    CompVel16
LD      0
ST      Out_VelCrescente

CompVel16:
LD      Array_Vel[17]
GE      Array_Vel[16]
JMPC    CompVel17
LD      0
ST      Out_VelCrescente

CompVel17:
LD      Array_Vel[18]
GE      Array_Vel[17]
JMPC    CompVel18
LD      0
ST      Out_VelCrescente

CompVel18:
LD      Array_Vel[19]
GE      Array_Vel[18]
JMPC    CompVel19
LD      0
ST      Out_VelCrescente

CompVel19:
LD      Array_Vel[20]
GE      Array_Vel[19]
JMPC    EndUpdateVel
LD      0
ST      Out_VelCrescente
EndUpdateVel:

(* LDN  Q06 *)
(* ST   Q06 *)
(* JMP  EndGetArrays *)

TTemp0:
LDN     TimerGetTemp.Q
JMPC    EndGetArrays
LD      0
ST      TrigGetTemp

EndGetArrays:
END_FUNCTION_BLOCK
```

Apêndice V.4 Função FCNCOMUN;

```
FUNCTION_BLOCK FCNCOMUN
VAR_INPUT
        In_SinalAtual : BYTE ;
END_VAR
VAR_OUTPUT
        Out_ComunicOK : BOOL ;
END_VAR
VAR
        UltSinal      : BYTE ;
        SinalOK       : BOOL ;

        DTempComunic : TIME := T#5s ;
        TimerComunic : TP;
        TrigComunic  : BOOL ;
END_VAR
CAL     TimerComunic(IN := TrigComunic,
                     PT := DTempComunic)
TComunic1:
LD      TimerComunic.Q
JMPC    TComunic0
LD      1
ST      TrigComunic
LD      SinalOK         (* Le o flag de comunicacao *)
ST      Out_ComunicOK   (* A cada DTempComunic atualiza status da comunicacao *)
R       SinalOK         (* Reseta o flag comprovante de comunicacao *)
JMP     FimVerificacao

TComunic0:
LDN     TimerComunic.Q
JMPC    FimVerificacao
LD      0
ST      TrigComunic
LD      UltSinal
EQ      In_SinalAtual   (* Compara sinal atual e ultimo sinal *)
JMPC    FimVerificacao (* Se ultimo sinal for igual ao atual, comunicacao pode
estar comprometida *)
LD      1
ST      SinalOK         (* Se ultimo sinal diferente do atual, comunicacao foi
comprovada *)
LD      In_SinalAtual   (* Atualiza o registro do ultimo sinal, caso sinal tenha
mudado *)
ST      UltSinal
FimVerificacao:
END_FUNCTION_BLOCK
```

Apêndice V.5 Função FCNINTRU;

```
FUNCTION_BLOCK FCNINTRU
VAR_INPUT
        In_Presenca : BOOL ;
        In_SalaHabilitada : BOOL ;
END_VAR
VAR_OUTPUT
        Out_SalaOcupada : BOOL ;
        Out_Intrusao : BOOL ;
        Out_Luzes : BOOL ;
END_VAR
(* Controle de acesso *)
LD      In_Presenca
JMPCN   SalaVazia

LD      In_SalaHabilitada
JMPCN   ExisteIntruso
S       Out_Luzes
S       Out_SalaOcupada
R       Out_Intrusao
JMP     EndContrAcesso

ExisteIntruso:
LD      1
```

```
        ST      Out_Intrusao
        ST      Out_SalaOcupada
        LD      0
        ST      Out_Luzes
        JMP     EndContrAcesso

        SalaVazia:
        LD      0
        ST      Out_SalaOcupada
        ST      Out_Intrusao
        ST      Out_Luzes

        EndContrAcesso:
        END_FUNCTION_BLOCK
```

Apêndice V.6 Função FCNLDTEMP;

```
        FUNCTION_BLOCK FCNLDTMP
        VAR_INPUT
                In_ComunicOK : BOOL ;
                In_SupervTemp : BYTE ;
                In_PotencTemp : WORD ;
        END_VAR
        VAR_OUTPUT
                Out_MaxTemp : UINT ;
                Out_MinTemp : UINT ;
        END_VAR
        VAR
                TempPotenc : UINT ;
                POTENCTEMP : TEMPCONV;
        END_VAR
        CarregaUserTemp:
        LD      In_ComunicOK
        JMPCN   NoComunic

                LD      In_SupervTemp (* Temperatura ajustada no supervisorio pelo usuario
em oC *)
                BYTE_TO_UINT
                ADD     1
                ST      Out_MaxTemp  (* Temperatura superior da faixa de conforto *)
                SUB     2
                ST      Out_MinTemp  (* Temperatura inferior da faixa de conforto *)
                JMP     EndCarregaUserTemp

        NoComunic:
                CAL     POTENCTEMP(TEMPWORD:= In_PotencTemp,
                                   TEMPMAX := 26,
                                   TEMPMIN := 20)
                LD      POTENCTEMP.TEMPGRAUS
                BYTE_TO_UINT
                ST      TempPotenc (* Temperatura ajustada pelo usuario no potenciometro do
CLP *)
                ADD     1
                ST      Out_MaxTemp (* Temperatura superior da faixa de conforto *)
                SUB     2
                ST      Out_MinTemp (* Temperatura inferior da faixa de conforto *)
        EndCarregaUserTemp:
        END_FUNCTION_BLOCK
```

Apêndice V.7 Função FCNVENTIL:

```
        FUNCTION_BLOCK FCNVENTIL
        VAR_INPUT
                In_Presenca : BOOL ;
                In_Temperatura : UINT ;
                In_MaxTemp : UINT ;
                In_MinTemp : UINT ;
                In_IncrementoPerc : BYTE ;
                In_VelocFixa : BYTE ;
```

```
        In_SalaHabilitada : BOOL ;
        In_VelocPerc : INT ;
END_VAR
VAR_IN_OUT
        Ext_VelocPerc : INT ;
END_VAR
VAR
        CounterFlag : BOOL ;
        Delay : TIME := T#10s ;
        Timer : TP ;

        Incremento    : INT;
        VelocFixaPerc : INT;
END_VAR
LD      In_IncrementoPerc
BYTE_TO_INT
ST      Incremento

LD      In_VelocFixa
BYTE_TO_INT
ST      VelocFixaPerc

(* Controle do ventilador *)
ContrVentilador:
CAL     Timer(IN:=CounterFlag,
              PT:=Delay)
LD      Timer.Q
EQ      1
R       CounterFlag
JMPC    AjustaVelocidadeVent

LD      In_SalaHabilitada
JMPCN   Desabilitada

LD      In_Presenca
JMPCN   VentSalaVazia

        LD      In_Temperatura
        LE      In_MaxTemp
        JMPC    ComparaMenor
        LD      Ext_VelocPerc
        ADD     Incremento
        ST      Ext_VelocPerc
        LD      1
        S       CounterFlag
        JMP     AjustaVelocidadeVent

        ComparaMenor:
        LD      In_Temperatura
        GE      In_MinTemp
        JMPC    AjustaVelocidadeVent
        LD      Ext_VelocPerc
        SUB     Incremento
        ST      Ext_VelocPerc
        LD      1
        S       CounterFlag
        JMP     AjustaVelocidadeVent

        VentSalaVazia:
        LD      VelocFixaPerc
        ST      Ext_VelocPerc
        JMP     AjustaVelocidadeVent

Desabilitada:
LD      VelocFixaPerc
ST      Ext_VelocPerc

AjustaVelocidadeVent:
LD      Ext_VelocPerc
LE      100
JMPC    Menor100
LD      100
ST      Ext_VelocPerc

Menor100:
```

```
LD      Ext_VelocPerc
GE      0
JMPC    PercOK
LD      0
ST      Ext_VelocPerc

PercOK:

END_FUNCTION_BLOCK
```

Apêndice V.8 Função SPDCONV;

```
FUNCTION_BLOCK SPDCONV
VAR_INPUT
        SPDPC : WORD ;
END_VAR
VAR_OUTPUT
        SPDBITS : WORD ;
END_VAR
VAR
        TEMPORARIO : UINT;
        DELTASPD : UINT := 100 ;
        VOLTMIN : UINT := 614 ;
        DELTABITS : UINT := 820 ;
END_VAR
LD      SPDPC
WORD_TO_UINT
DIV     2
MUL     DELTABITS
DIV     DELTASPD
MUL     2
ST      TEMPORARIO
LD      VOLTMIN
ADD     TEMPORARIO
UINT_TO_WORD
ST      SPDBITS
END_FUNCTION_BLOCK
```

Apêndice V.9 Função TEMPCONV;

```
FUNCTION_BLOCK TEMPCONV
VAR_INPUT
        TEMPWORD : WORD ;   (* Entrada em WORD *)
        TEMPMIN : UINT;
        TEMPMAX : UINT;
END_VAR
VAR_OUTPUT
        TEMPGRAUS : BYTE ;   (* Saida em graus Celsius*)
END_VAR
VAR
        TEMPORARIO : UINT;
        DELTATEMP : UINT ;
        MAXBITS: UINT := 1023;
END_VAR
LD      TEMPMAX
SUB     TEMPMIN
ST      DELTATEMP

LD      TEMPWORD
WORD_TO_UINT
MUL     DELTATEMP
DIV     MAXBITS
ADD     TEMPMIN
ST      TEMPORARIO
LD      TEMPORARIO
UINT_TO_BYTE
```

```
ST      TEMPGRAUS
END_FUNCTION_BLOCK
```

Apêndice VI. Listagem do Programa Supervisório

O programa supervisório foi desenvolvido em ambiente LabWindows, desenvolvido pela National Instruments®. Este ambiente prevê a utilização da liguagem C, e disponibiliza controles para uso em ambiente Windows (como telas, botões, gráficos, etc..).

O programa em LabWindows é composto de alguns arquivos:

- **.UIR** - guarda as informações da tela de interface (telas, botões, etc);
- **.H** - como é um programa em C, os arquivos .h são os arquivos *header*, que contém os *prototypes* das funções utilizadas;
- **.C** - contém o programa propriamente dito.

Apêndice VI.1 Supervisor.uir

Este programa contém a interface do programa (todas as telas). A listagem desta parte não é possível de ser apresentada, pois o LabWindows grava esse arquivo em formato próprio.

A saída deste programa são as telas do programa, já apresentadas em item anterior.

Apêndice VI.2 Supervisor.h

Este arquivo contém as definições das funções utilizadas no programa (como as chamadas aos item da tela e das funções de *callback* (chamadas quando o usuário interage com os comandos disponibilizados pelo programa).

```
/*****************************************************************************/
/* LabWindows/CVI User Interface Resource (UIR) Include File                 */
/* Copyright (c) National Instruments 1998. All Rights Reserved.             */
/*                                                                           */
/* WARNING: Do not add to, delete from, or otherwise modify the contents     */
/*          of this include file.                                            */
/*****************************************************************************/

#include <userint.h>
```

```
#ifdef __cplusplus
    extern "C" {
#endif

     /* Panels and Controls: */

#define  PANEL_3                        1
#define  PANEL_3_COMMANDBUTTON          2       /* callback function:
PanelDiscard */
#define  PANEL_3_STRIPCHART             3

#define  PNL_CTRL                       2       /* callback function: MainControl
*/
#define  PNL_CTRL_LED_FIRE2             2
#define  PNL_CTRL_LED_SMOKE2            3
#define  PNL_CTRL_LED_PRESENCE2         4
#define  PNL_CTRL_NSL_SPEED2            5
#define  PNL_CTRL_TBT_EMERGENCY2        6
#define  PNL_CTRL_NTH_TEMP2             7
#define  PNL_CTRL_COMMANDBUTTON_2       8
#define  PNL_CTRL_TBT_COMMSTATUS2       9
#define  PNL_CTRL_LED_FIRE1             10
#define  PNL_CTRL_LED_SMOKE1            11
#define  PNL_CTRL_LED_PRESENCE1         12
#define  PNL_CTRL_NSL_SPEED1            13
#define  PNL_CTRL_TBT_NORMALSIT2        14
#define  PNL_CTRL_TBT_NORMALSIT1        15
#define  PNL_CTRL_TBT_EMERGENCY1        16
#define  PNL_CTRL_NTH_TEMP1             17
#define  PNL_CTRL_CMB_SHOWHIST          18      /* callback function: ShowHist */
#define  PNL_CTRL_TBT_COMMSTATUS1       19
#define  PNL_CTRL_CMB_QUIT              20      /* callback function: Quit */
#define  PNL_CTRL_LED_INTRUSION2        21
#define  PNL_CTRL_LED_INTRUSION1        22
#define  PNL_CTRL_CMB_CONNECT           23      /* callback function: DDEConnect
*/
#define  PNL_CTRL_CMB_RESETFIRE2        24      /* callback function: ResetFire
*/
#define  PNL_CTRL_CMB_RESETFIRE1        25      /* callback function: ResetFire
*/
#define  PNL_CTRL_TEXTMSG               26
#define  PNL_CTRL_DECORATION_2          27
#define  PNL_CTRL_DECORATION            28
#define  PNL_CTRL_TEXTMSG_2             29

#define  PNL_GRAPH1                     3
#define  PNL_GRAPH1_CMB_CLOSEGRAPH1     2       /* callback function: CloseGraph
*/
#define  PNL_GRAPH1_NUM_REFRESHRATE     3
#define  PNL_GRAPH1_GRAPHTEMP1          4
#define  PNL_GRAPH1_GRAPHFANSPEED1      5

#define  PNL_USER                       4       /* callback function: MainControl
*/
#define  PNL_USER_NUM_INCSPEED2         2
#define  PNL_USER_NUM_INITSPEED2        3
#define  PNL_USER_TBT_ENABLED2          4
#define  PNL_USER_KNB_USERTEMP2         5
#define  PNL_USER_NUM_INCSPEED1         6
#define  PNL_USER_NUM_INITSPEED1        7
#define  PNL_USER_TBT_ENABLED1          8
#define  PNL_USER_KNB_USERTEMP1         9
#define  PNL_USER_DECORATION            10
#define  PNL_USER_DECORATION_2          11
#define  PNL_USER_TEXTMSG               12
#define  PNL_USER_TEXTMSG_2             13

     /* Menu Bars, Menus, and Menu Items: */

          /* (no menu bars in the resource file) */

     /* Callback Prototypes: */
```

```
        int  CVICALLBACK CloseGraph(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
        int  CVICALLBACK DDEConnect(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
        int  CVICALLBACK MainControl(int panel, int event, void *callbackData, int
eventData1, int eventData2);
        int  CVICALLBACK PanelDiscard(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
        int  CVICALLBACK Quit(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
        int  CVICALLBACK ResetFire(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
        int  CVICALLBACK ShowHist(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);


        #ifdef __cplusplus
            }
        #endif
```

Apêndice VI.3 Defines.h

Este arquivo foi gerado para guardar as constantes utilizadas pelo programa. Contém também os endereços das variáveis de memória do CLP, que serão utilizadas na troca de informações entre o CLP e o programa supervisório.

```
        // Número de pontos nos históricos:
        #define ARRAY_SIZE     255

        //  Constantes de Ventilação
        #define LOWTEMPTOL 1          //  Limite inferior da faixa de temperatura
        #define HIGHTEMPTOL 1         //  Limite superior da faixa de temperatura
        #define FIRETEMP 55                 //  Temperatura de incêndio (Celsius)

        //      Constantes do controle de temperatura:
        #define EVENT_RATE     1000
        #define DELAY_TIME     5

        #define UNOCCUPIED_LEVEL 20
        #define SPEED_STEP     5
        #define MAX_SPEED      100
        #define MIN_SPEED      0

        // Constantes de DDE
        // Constantes de Conexão
        #define NAME                    "DDE"
        #define TOPIC                   "SUCOMA"
        #define DDE_TIMEOUT     50

        //      Constantes de Escrita
        #define itmWriteHabilitada0           "MW4"          // Item DDE para Escrita do
Sinal de Habilitação da Sala
        #define itmWriteEmergencia0           "MW6"          // Item DDE para Escrita do
Sinal de Emergencia
        #define itmWriteUsuarioTemp0  "MW8"          // Item DDE para Escrita do Sinal de
Temperatura Ajusta pelo Usuário
        #define itmWriteCommSinal             "MW10"         // Item DDE para Escrita do
Sinal de Comunicação
        #define itmWriteSitNormal0            "MW12"         // Item DDE para Escrita do
Sinal de Situação Normalizada
        #define itmWriteIncendio0             "MW14"         // Item DDE para Escrita do
Sinal de Incendio

        //  Constantes do Leitura
```

```
        #define itmReadIncendio0              "MB20"          // Item DDE para Leitura do
sinal de incendio
        #define itmReadIntrusao0              "MB22"          // Item DDE para Leitura do
sinal de intrusao
        #define itmReadSalaOcupada0           "MB24"          // Item DDE para Leitura do
sinal de sala ocupada
        #define itmReadTempAtual0             "MB26"          // Item DDE para Leitura do
sinal de temperatura na sala
        #define itmReadVelAtual0              "MB28"          // Item DDE para Leitura do
sinal de velocidade do ventilador
        #define itmReadSmoke0                 "MB30"          // Item DDE para Leitura do
sinal de fumaça

        // Constantes de Escrita/Leitura
        #define itmUserIncSpeed0              "MW16"          // Item DDE para
Leitura/Escrita do Incremento de Velocidade
        #define itmUserVelFixa0                       "MW18"          // Item DDE para
Leitura/Escrita da Velocidade Fixa
```

Apêndice VI.4 Functions.h

Contém os *prototypes* das funções utilizadas no programa.

```
        int CVICALLBACK ClientDDECallback (unsigned int handle, char *topicName, char
*itemName, int event,
            int dataFmt, int dataSize, void *dataPtr, void *callbackData);


        int Initialize_DDE (void);

        int     CVICALLBACK Generate_Step (int panel, int controlID, int event,
                                                                    void *callbackData,
int eventData1, int eventData2);


        int     VerifyTemperature(double, double);
        int             IncreaseSpeedOneLevel(double*);
        int             DecreaseSpeedOneLevel(double*);
        int             SetSpeedToLevel(double*, double);
        int     CheckSpeed(double*);
        int     Initialize_Variables (void);
        void    IncluiArray(int*, int);
        int     Initialize_DDE (void);
```

Apêndice VI.5 Gblvars.h

Este arquivo contém as definições das variáveis globais.

```
        // Entradas do usuário:
        float   gblUserTemp[2];               // Temperatura da sala definida pelo usuário
        int             gblUserIncSpeed[2]; // Incremento da velocidade do ventilador
        int     gblUserVelFixa[2];  // Velocidade do ventilador quando não há presença na
sala
        int UserResetFire;                    // Reseta o valor de Incendio na planta

        // Entradas do controle:
        int     gblEnabled[2];                // Indica se a sala está habilitada ou
desabilitada
        int     gblEmergency[2];              // Indica se o edificio está em situação de
emergencia ou não
        int             gblNormalSit[2];              // Indica se o estado de situação
normal foi reestabelecido
        int             gblIsFire[2];                 // Indica se o estado atual é de
incêndio
```

```
        // Sinal de comunicação:
        int             CommSinal;                      // Sinal enviado para coprovação de
comunicação
        BOOL    ConnectionOK;           // Indica que existe uma conexão DDE ativa

        // Estado da planta:
        int     gblFire[2];                     // Indica se há indicios de incendio na sala
        int     gblIntrusion[2];                // Indica intrusão na sala
        int     gblOccupancy[2];                // Indica presença na sala
        float   gblCurrentTemp[2];              // Temperatura atual da sala
        int             gblFanSpeed[2];                 // Velocidade do ventilador
        int             gblSmoke[2];

        int             gblAlarme[2];
        int             gblLuzes[2];


        // Arrays de histórico:
        int     TempArray[2][ARRAY_SIZE+1];
        int     VelArray[2][ARRAY_SIZE+1];


        // Outras Variáveis
        clock_t start,finish;           // Variáveis de tempo, para atualização do Histórico
```

Apêndice VI.6 Supervisor.c

Este é o arquivo que contém o programa propriamente dito.

```
        #include <utility.h>
        #include <ansi_c.h>

        #include <formatio.h>
        #include <cvirte.h>             /* Needed if linking in external compiler; harmless
otherwise */
        #include <userint.h>
        #include <ddesupp.h>    /* Constantes e funções para suporte a DDE */

        #include "defines.h"    /* Definições de constantes usadas no programa */
        #include "functions.h" /* Declarações das funções utilizadas no programa */
        #include "gblvars.h"    /* Declarações das variáveis globais */
        #include "supervisor.h"

        static int pnlControl, pnlUser, pnlGraph1;
        static unsigned DDEConnID;


        int main (int argc, char *argv[])
        {
                if (InitCVIRTE (0, argv, 0) == 0)       /* Needed if linking in external
compiler; harmless otherwise */
                        return -1;      /* out of memory */
                if ((pnlControl = LoadPanel (0, "supervisor.uir", PNL_CTRL)) < 0)
                        return -1;
                if ((pnlUser = LoadPanel (0, "supervisor.uir", PNL_USER)) < 0)
                        return -1;
                if ((pnlGraph1 = LoadPanel (0, "supervisor.uir", PNL_GRAPH1)) < 0)
                        return -1;
                DisplayPanel (pnlUser);
                DisplayPanel (pnlControl);
                InitializeVariables();
                InstallMainCallback ( Generate_Step , 0 , 1 );
                SetIdleEventRate ( EVENT_RATE );
                RunUserInterface ();
                return 0;
        }
```

```
int CVICALLBACK MainControl (int panel, int event, void *callbackData,
                int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_GOT_FOCUS:

                        break;
                case EVENT_LOST_FOCUS:

                        break;
                case EVENT_CLOSE:
                        QuitUserInterface(0);
                        break;
                }
        return 0;
}


int CVICALLBACK Generate_Step (int panel, int controlID, int event,
                                                        void *callbackData, int
eventData1,
                                                        int eventData2)

{
        int     TempStatus[2];
        int             DDEStatus;
        char    buffer[10];
        int             IntCurrentTemp;
        double  duration;
        int             RefreshTime;

        switch (event)
                {
                case EVENT_IDLE:
                if(ConnectionOK==1)
                        {

                        // Desliga envio de mensagens de erro para a tela:
                        // Erros serão tratados aqui mesmo
                        DisableBreakOnLibraryErrors ();

                        // Envia sinal de comprovação de comunicação:
                                if(CommSinal==2)
                                  {
                                  CommSinal = 3;
                                  sprintf(buffer, "%d", CommSinal);
                                DDEStatus = ClientDDEWrite(DDEConnID,
itmWriteCommSinal, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                  {
                                  ConnectionOK = 0;
                                  }
                                    }
                                  else
                                    {
                                    CommSinal = 2;
                                  sprintf(buffer, "%d", CommSinal);
                                    DDEStatus = ClientDDEWrite(DDEConnID,
itmWriteCommSinal, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                  {
                                  ConnectionOK = 0;
                                  }
                                    }

                              // Le variaveis de estado do CLP:

                                    // Fumaça
                                    if(ConnectionOK==1)
                                            {
                                            DDEStatus = ClientDDERead(DDEConnID,
itmReadSmoke0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                            if(DDEStatus!=10)
                                                    {
```

```
                                        ConnectionOK = 0;
                                        }
                                else
                                        {
                                        sscanf(buffer, "%d", &gblSmoke[0]);
                                        }
                                }

                        // Incendio
                        if(ConnectionOK==1)
                                {
                                DDEStatus = ClientDDERead(DDEConnID,
itmReadIncendio0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                else
                                        {
                                        sscanf (buffer, "%d", &gblFire[0]);
                                        }
                                }

                        // Intrusão
                        if(ConnectionOK==1)
                                {
                                DDEStatus = ClientDDERead(DDEConnID,
itmReadIntrusao0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                else
                                        {
                                        sscanf (buffer, "%d",
&gblIntrusion[0]);
                                        }
                                }

                        // Presença
                        if(ConnectionOK==1)
                                {
                                DDEStatus = ClientDDERead(DDEConnID,
itmReadSalaOcupada0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                else
                                        {
                                        sscanf (buffer, "%d",
&gblOccupancy[0]);
                                        }
                                }

                        // Temperatura Atual
                        if(ConnectionOK==1)
                                {
                                DDEStatus = ClientDDERead(DDEConnID,
itmReadTempAtual0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                else
                                        {
                                        sscanf (buffer, "%d",
&IntCurrentTemp);
                                        }
                                }
                        gblCurrentTemp[0] = IntCurrentTemp;

                        // Velocidade Atual do Ventilador (em porcentagem)
                        if(ConnectionOK==1)
                                {
```

```
                                        DDEStatus = ClientDDERead(DDEConnID,
itmReadVelAtual0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                        if(DDEStatus!=10)
                                                {
                                                ConnectionOK = 0;
                                                }
                                        else
                                                {
                                                sscanf (buffer, "%d",
&gblFanSpeed[0]);
                                                }
                                        }

                                // Valor do incremento da velocidade do ventilador
(em porcentagem)
                                if(ConnectionOK==1)
                                        {
                                        DDEStatus = ClientDDERead(DDEConnID,
itmUserIncSpeed0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                        if(DDEStatus!=10)
                                                {
                                                ConnectionOK = 0;
                                                }
                                        else
                                                {
                                                sscanf (buffer, "%d",
&gblUserIncSpeed[0]);
                                                }
                                        }

                                // Valor da velocidade para sala desocupada
                                if(ConnectionOK==1)
                                        {
                                        DDEStatus = ClientDDERead(DDEConnID,
itmUserVelFixa0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                        if(DDEStatus!=10)
                                                {
                                                ConnectionOK = 0;
                                                }
                                        else
                                                {
                                                sscanf (buffer, "%d",
&gblUserVelFixa[0]);
                                                }
                                        }

                        // Le entradas do usuário:
                                GetCtrlVal (pnlControl, PNL_CTRL_TBT_EMERGENCY1,
&gblEmergency[0]);
                                GetCtrlVal (pnlControl, PNL_CTRL_TBT_NORMALSIT1,
&gblIsFire[0]);
                                GetCtrlVal (pnlUser, PNL_USER_TBT_ENABLED1,
&gblEnabled[0]);
                                GetCtrlVal (pnlUser, PNL_USER_KNB_USERTEMP1,
&gblUserTemp[0]);
                                GetCtrlVal (pnlUser, PNL_USER_NUM_INCSPEED1,
&gblUserIncSpeed[0]);
                                GetCtrlVal (pnlUser, PNL_USER_NUM_INITSPEED1,
&gblUserVelFixa[0]);
                                GetCtrlVal (pnlGraph1, PNL_GRAPH1_NUM_REFRESHRATE,
&RefreshTime);


                        // Verifica estado da temperatura:
                                TempStatus[0]=VerifyTemperature(gblCurrentTemp[0],
gblUserTemp[0]);

                        // Atualiza saida para o CLP:
                                if(ConnectionOK==1)
                                        {
                                        sprintf(buffer, "%d", (int)gblUserTemp[0]);
                                        DDEStatus = ClientDDEWrite(DDEConnID,
itmWriteUsuarioTemp0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                        if(DDEStatus!=10)
                                                {
```

```
                                        ConnectionOK = 0;
                                        }
                                }

                        if(ConnectionOK==1)
                                {
                                sprintf(buffer, "%d", (int)gblUserIncSpeed[0]);
                                DDEStatus = ClientDDEWrite(DDEConnID,
itmUserIncSpeed0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                }

                        if(ConnectionOK==1)
                                {
                                sprintf(buffer, "%d", (int)gblUserVelFixa[0]);
                                DDEStatus = ClientDDEWrite(DDEConnID,
itmUserVelFixa0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                }

                        if(ConnectionOK==1)
                                {
                                sprintf(buffer, "%d", gblEnabled[0]);
                                DDEStatus = ClientDDEWrite(DDEConnID,
itmWriteHabilitada0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                }

                        if(ConnectionOK==1)
                                {
                                sprintf(buffer, "%d", gblEmergency[0]);
                                DDEStatus = ClientDDEWrite(DDEConnID,
itmWriteEmergencia0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                }

                        if(ConnectionOK==1)
                                {
                                sprintf(buffer, "%d", gblNormalSit[0]);
                                DDEStatus = ClientDDEWrite(DDEConnID,
itmWriteSitNormal0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                }

                        if(ConnectionOK==1)
                                {
                                sprintf(buffer, "%d", gblIsFire[0]);
                                DDEStatus = ClientDDEWrite(DDEConnID,
itmWriteIncendio0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                if(DDEStatus!=10)
                                        {
                                        ConnectionOK = 0;
                                        }
                                }

                        // Caso o usuário tenha clicado o botão Reset Fire, essa
informação será enviada agora ao Supervisório

                        if(ConnectionOK==1)
                                {
```

```
                                        sprintf(buffer, "%d", UserResetFire);
                                    DDEStatus = ClientDDEWrite(DDEConnID,
itmWriteSitNormal0, CF_TEXT, buffer, sizeof(buffer), DDE_TIMEOUT);
                                        if(DDEStatus!=10)
                                                {
                                                ConnectionOK = 0;
                                                }
                                        }

                                //  Reset o valor de ResetFire - deve ser 1 apenas para ser
enviada
                                    UserResetFire=0;

                                // Atualiza saida para o usuario:
                                        SetCtrlVal(pnlControl, PNL_CTRL_NSL_SPEED1,
gblFanSpeed[0]);                // Atualiza velocidade do ventilador
                                        SetCtrlVal(pnlControl, PNL_CTRL_NTH_TEMP1,
gblCurrentTemp[0]);     // Atualiza temperatura da sala
                                        SetCtrlVal(pnlControl, PNL_CTRL_LED_PRESENCE1,
gblOccupancy[0]);               // Atualiza presença na sala
                                        SetCtrlVal(pnlControl, PNL_CTRL_LED_SMOKE1,
gblSmoke[0]);           // Atualiza fumaça na sala
                                        SetCtrlVal(pnlControl, PNL_CTRL_LED_FIRE1,
gblFire[0]);                    // Atualiza indícios de incêndio na sala
                                        SetCtrlVal(pnlControl, PNL_CTRL_LED_INTRUSION1,
gblIntrusion[0]);                       // Atualiza indícios de incêndio na sala

                                // Atualiza arrays para a construção do gráfico:
                                        finish=clock();
                                        duration=(double)(finish-start)/CLOCKS_PER_SEC;
                                        if (duration > RefreshTime)
                                                {
                                                IncluiArray(TempArray[0], gblCurrentTemp[0]);
                                                IncluiArray(VelArray[0], gblFanSpeed[0]);
                                                PlotStripChart (pnlGraph1,
PNL_GRAPH1_GRAPHTEMP1 , gblCurrentTemp, 1, 0, 0, 3);
                                                PlotStripChart (pnlGraph1,
PNL_GRAPH1_GRAPHFANSPEED1 , gblFanSpeed, 1, 0, 0, 1);
                                                }
                                if(ConnectionOK == 0)
                                        {
                                        MessagePopup ("Supervisor", "Erro de
comunicação!!!");

                                        SetCtrlVal(pnlControl, PNL_CTRL_CMB_CONNECT,0);

                                        Initialize_Variables();

                                        SetCtrlVal(pnlControl, PNL_CTRL_NSL_SPEED1,
gblFanSpeed[0]);                // Atualiza velocidade do ventilador
                                        SetCtrlVal(pnlControl, PNL_CTRL_NTH_TEMP1,
gblCurrentTemp[0]);     // Atualiza temperatura da sala
                                        SetCtrlVal(pnlControl, PNL_CTRL_LED_PRESENCE1,
gblOccupancy[0]);               // Atualiza presença na sala
                                        SetCtrlVal(pnlControl, PNL_CTRL_LED_SMOKE1,
gblSmoke[0]);           // Atualiza fumaça na sala
                                        SetCtrlVal(pnlControl, PNL_CTRL_LED_FIRE1,
gblFire[0]);                    // Atualiza indícios de incêndio na sala
                                        SetCtrlVal(pnlControl, PNL_CTRL_LED_INTRUSION1,
gblIntrusion[0]);                       // Atualiza indícios de incêndio na sala


                                        }
                                // Reabilita envio de mensagens de erro para a tela:
                                EnableBreakOnLibraryErrors ();
                                }
                                break;
                        }
                        return 0;
        }

        int Initialize_Variables (void)
        {
        short i;
```

```
        for(i=0; i<2; i++)
              {
              gblAlarme[i] = 0;
              gblLuzes[i] = 0;
              gblOccupancy[i]  = 0;
              gblSmoke[i] = 0;

              gblFanSpeed[i] = 0;
              }

        ConnectionOK=0;
        SetCtrlVal(pnlControl, PNL_CTRL_TBT_COMMSTATUS1,1);
        SetCtrlVal(pnlControl, PNL_CTRL_TBT_COMMSTATUS2,1);

        GetCtrlVal (pnlControl, PNL_CTRL_TBT_EMERGENCY1, &gblEmergency[0]);
        GetCtrlVal (pnlUser, PNL_USER_TBT_ENABLED1, &gblEnabled[0]);
        GetCtrlVal (pnlUser, PNL_USER_KNB_USERTEMP1, &gblUserTemp[0]);

        CommSinal = 2;

        return 0;
      }

      int Initialize_DDE (void)
      {

            int DDE_Error;
            int DDE_Server_Handle;
            /* establishes connection with server */

            DisableBreakOnLibraryErrors ();
            DDE_Error=ConnectToDDEServer (&DDEConnID, NAME, TOPIC, ClientDDECallback,
0);

            // Verifica se o DDEServer response. Em caso contrário, inicializa o
DDEServer
            if (DDE_Error == -26)
                  {
                  LaunchExecutableEx
("C:\\Users\\Sinted\\DRIVER~1\\ARQUIVOS\\Dde.exe",LE_SHOWMINIMIZED,&DDE_Server_Handle);
                  ConnectToDDEServer (&DDEConnID, NAME, TOPIC, ClientDDECallback, 0);
                  }

            ConnectToDDEServer (&DDEConnID, NAME, TOPIC, ClientDDECallback, 0);
            EnableBreakOnLibraryErrors ();

            return 1;
      }

      int CVICALLBACK ClientDDECallback (unsigned int handle, char *topicName, char
*itemName, int event,
          int dataFmt, int dataSize, void *dataPtr, void *callbackData)
      {
            unsigned int   value;

            switch (event) {
                /*
                case DDE_DATAREADY:
                            if  (stromp(itemName, itmReadOccupancy) == 0)
                                  {
                                  sscanf ((char*)dataPtr, "%i", &value);
                                  gblOccupancy = value;
                                  }

                            if  (stromp(itemName, itmReadSmoke ) == 0)
                                  {
                                  sscanf ((char*)dataPtr, "%i", &value);
                                  gblSmoke = value;
                                  }

                            if  (stromp(itemName, itmReadTemp ) == 0)
                                  {
                                  sscanf ((char*)dataPtr, "%i", &value);
                                  gblCurrentTemp = value;
                                  }
```

```
                */
            break;


            case DDE_DISCONNECT:
                /*
                SetCtrlVal (panel2, PANEL_2_TEXTBOX, "Connection Closed!!!\n");
                */
            break;
        }
        return 0;
}

int VerifyTemperature (double CurrentTemp, double UserTemp)

{
        if (CurrentTemp<UserTemp-LOWTEMPTOL)
                {
                return -1;
                }
        if (UserTemp-LOWTEMPTOL<=CurrentTemp && CurrentTemp<=UserTemp+HIGHTEMPTOL)
                {
                return 0;
                }
        if (CurrentTemp>UserTemp+HIGHTEMPTOL && CurrentTemp<FIRETEMP)
                {
                return 1;
                }
        if (CurrentTemp>=FIRETEMP)
                {
                return 2;
                }
        return 0;
}

int IncreaseSpeedOneLevel(double* Speed)
{
        *Speed += SPEED_STEP;
        CheckSpeed(Speed);
        return 0;
}

int DecreaseSpeedOneLevel(double* Speed)
{
        *Speed -= SPEED_STEP;
        CheckSpeed(Speed);
        return 0;
}

int SetSpeedToLevel(double* Speed, double NewSpeed)
{
        *Speed = NewSpeed;
        CheckSpeed(Speed);
        return 0;
}

int CheckSpeed(double* Speed)
{
        if(*Speed > MAX_SPEED)
                {
                *Speed = MAX_SPEED;
                }

        if(*Speed < MIN_SPEED)
                {
                *Speed = MIN_SPEED;
                }
        return 0;
}

void IncluiArray(int* Vetor, int NovoElemento)
{
        short i;
        start=clock();
        for(i=0; i<ARRAY_SIZE-1; i++)
```

```
                {
                Vetor[i]=Vetor[i+10];
                }
        Vetor[ARRAY_SIZE] = NovoElemento;
}


int CVICALLBACK ResetFire (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {

                case EVENT_COMMIT:
                                UserResetFire=1;
                        break;
                }
        return 0;
}


int CVICALLBACK PanelDiscard (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:

                        break;
                }
        return 0;
}

int CVICALLBACK ShowHist (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                                DisplayPanel(pnlGraph1);
                        break;
                }
        return 0;
}

int CVICALLBACK Quit (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                                QuitUserInterface(0);
                        break;
                }
        return 0;
}

int CVICALLBACK DDEConnect (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)

{
        int ToggleConn;
        switch (event)
                {
                case EVENT_COMMIT:

                        GetCtrlVal (pnlControl, PNL_CTRL_CMB_CONNECT, &ToggleConn);
                        if (ToggleConn==1)
                                {
                                if(Initialize_DDE())
                                        {
                                        SetCtrlVal(pnlControl,
PNL_CTRL_TBT_COMMSTATUS1,0);
                                        SetCtrlVal(pnlControl,
PNL_CTRL_TBT_COMMSTATUS2,0);
```