ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

DEPARTAMENTO DE ENGENHARIA MECATRÔNICA E DE SISTEMAS MECÂNICOS

PMC 581 – Projeto Mecânico II

Sistemas de Informação para Automação Predial

Relatório Final

Prof. Orientador: José Reinaldo Silva

Aluno:	Número Usp
Marco Dyodi Takahashi	**2368088**
Patrick M. von Schaaffhausen	**2238621**

São Paulo
2001

ÍNDICE

Índice de Figuras

1 Introdução

Atualmente a integração de sistemas é uma estratégia capaz de garantir uma melhoria significativa no uso dos recursos energéticos, devido o aumento da eficiência da coordenação das funções realizadas pelas diversas partes de um sistema resultando em uma melhora qualitativa e quantitativa do funcionamento do sistema como um todo.

No entanto, em grande parte dos "edifícios inteligentes" atualmente em funcionamento, o que se observa são sistemas em que não há uma integração entre os diversos sub-sistemas que o compõe não havendo assim um compartilhamento das informações.

Nossa proposta é apresentar um modelo de controle predial integrando as diversas partes do sistema com o objetivo de diminuir os custos de operação, diminuir o gasto energético, aumentar a vida útil do sistema e garantir seu uso com conforto e segurança.

Em um sistema integrado os dados de diversos sub-sistemas são compartilhados. Por exemplo, o sub-sistema de iluminação e de presença, apesar de possuírem finalidades distintas, possuem os mesmos dados de entrada. Em sistemas não integrados estes dados são coletados independentemente; aumentando o custo do sistema, aumentando os requisitos computacionais e gerando margem para inconsistência entre as informações. Já em sistemas integrados, como o proposto, esses problemas e inconsistências são eliminados. Além de permitir uma maior flexibilidade, possibilitando alterações e novas implementações com maior facilidade.

2 Sistema de Controle

Neste trabalho, construiremos uma maquete com três andares, onde se deseja monitorar e controlar os sistemas de iluminação, segurança, conforto térmico e elevador. Ao todo são cinco ambientes, um no piso térreo e mais dois ambientes no primeiro e segundo pisos.

Serão instalados sensores de presença, temperatura e de fumaça, além de iluminação e ventiladores para o controle térmico, conforme o diagrama.

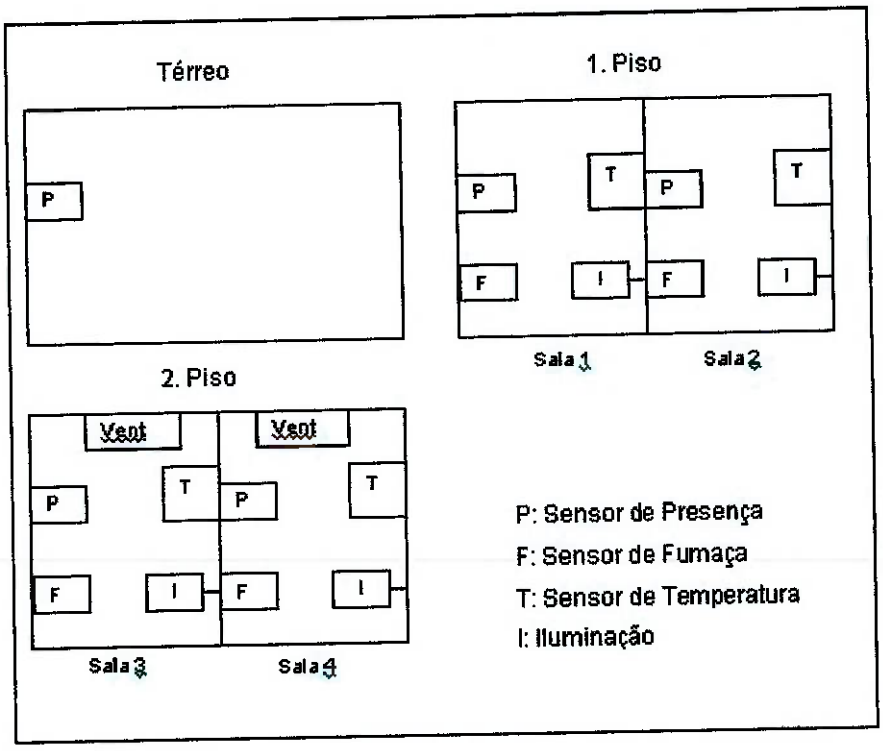

Figura 2-1: Esquema dos sensores na maquete

Este sistema será implementado em uma maquete conforme o modelo a seguir:

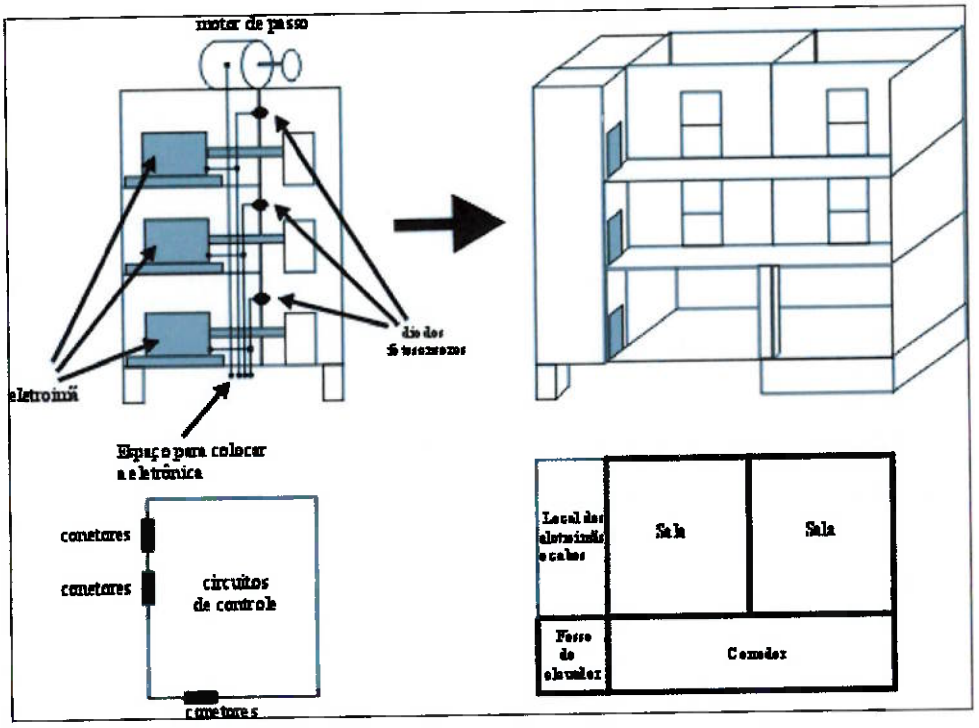

Figura 2-2: Esquema da Maquete

A maquete projetada conforme a figura acima pode ser vista na foto abaixo:

Figura 2-3: Foto da maquete

A seguir será apresentada uma descrição dos diferentes subsistemas, que serão tratados neste trabalho, com sua modelagem em Redes de Petri feitas através do programa *Visual Object 2.0.*

2.1 *Sistema de Segurança*

O sistema de segurança é responsável pela verificação da ocorrência de situações não esperadas e que possam afetar a integridade física das pessoas, das instalações presentes, ou interferência com os processos inerentes aos usuários do prédio (obtenção indevida de informação, etc.)

Em nosso projeto teremos duas preocupações referentes à segurança que serão divididos em dois sub-sistemas: o controle de acesso aos ambientes e a detecção de incêndio.

Quanto ao controle de acesso, os ambientes podem estar habilitados ou não para o uso, conforme o plano da administração do prédio. Estando desabilitados, a detecção de presença será tratada como uma intrusão e esta informação deverá ser passada para o sistema supervisor que tomará as providências necessárias na sala em que se encontra a situação detectada. Isto permite uma certa flexibilidade ao sistema, conforme os recursos disponíveis. O supervisor pode ainda, fazer verificações adicionais para melhor definir o que está acontecendo, podendo acionar um sistema de alarme, alertar um serviço de vigilância conectado a uma central de policia, ou tomar medidas no sentido de isolar os ambientes que se encontram na vizinhança do ambiente invadido.

2.1.1 *Modelagem do sistema de detecção de intrusão:*

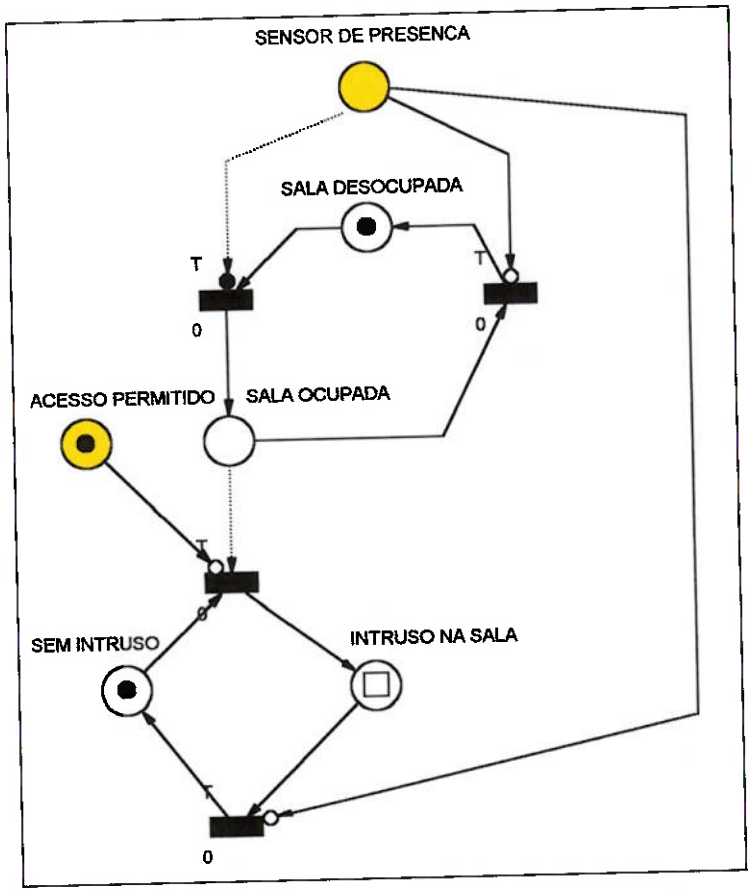

Figura 2-4:Rede de controle de detecção de presença

Para este modelo tem-se como parâmetro definido pelo usuário, a administração do prédio: a política de acesso, isto é, a permissão ou não do acesso naquele momento. Esta permissão está representada na rede GHENeSys (General Hierarchical Enhanced Net System) da figura 2-4 através de um pseudo-box (em amarelo). Os pseudo-boxes representam elementos cuja marcação não está no controle do sub-sistema em questão, mas sim de elementos externos (outros sub-sistemas, usuários externos, ou sinais originários do mundo externo ao sub-sistema). Na rede GHENeSys os pseudo-boxes são usados para representar eventos previsíveis porém não controláveis, por exemplo, a chegada de um indivíduo ao ambiente (elemento associado ao sensor de presença na figura 2-4).

Os dois estados possíveis para este modelo são: sala ocupada ou desocupada, habilitado a partir do sinal proveniente do sensor de presença.

A situação de incêndio pode ser detectada com os seguintes eventos: detecção de fumaça na sala e detecção de foco de temperatura elevada. Essa detecção funciona da seguinte forma: em presença de fumaça o sistema começa a estudar o comportamento da temperatura do local nos últimos dez minutos, dessa forma, mantida a temperatura elevada a hipótese de incêndio pode ser confirmada. Confirmado o foco de incêndio o sistema procura contactar o supervisor para que as medidas adequadas sejam tomadas. Em caso de impossibilidade de comunicação com o supervisor os *sprinklers* são acionados.

Ao receber a informação de incêndio o supervisor pode tanto confirmar a existência de incêndio, o que acionará os *sprinklers* imediatamente, ou negá-la, informando que as condições estão normais. Neste caso o sistema de emergência é desativado.

Os sprinklers também podem ser acionados manualmente, caso isso seja necessário por questões emergenciais.

2.1.2 Modelagem do sistema de detecção de incêndio.

Neste modelo os parâmetros em que pode haver interferência do sistema supervisório são

- Condições normais
- Confirmação de incêndio
- Atuação manual dos sprinkler

As entradas do sistema são:

- Sensor de fumaça
- Sensor de temperatura

Os estados possíveis são:

- Emergência
- Normal

As saídas são:

- Atuação nos *sprinklers*
- Comunicação de incêndio para sistema supervisório

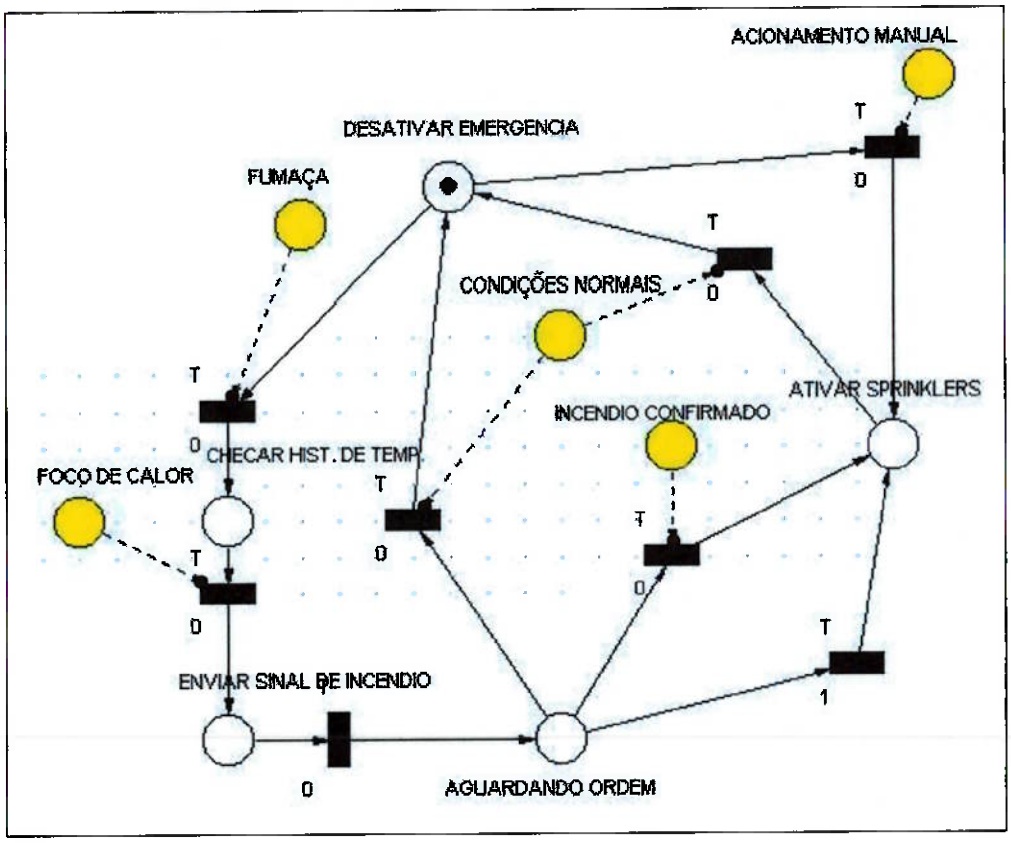

Figura 2-5: Rede de controle de detecção de incêndio

2.2 Sistema de Iluminação

O acionamento da iluminação das salas será permitido caso o uso da sala esteja habilitado. A habilitação ou não da sala é feita pelo sistema supervisor, de acordo com a política de ocupação do edifício. Estando habilitada o usuário terá a possibilidade de acendê-la através do interruptor. Uma vez acesa, a luz se apagará ou através do interruptor ou se não for mais detectada presença na sala, quando a iluminação retorna ao seu estado *default*, que é apagada.

2.2.1 Modelagem do sistema de iluminação:

Parâmetros:

- Habilitação das luzes

Entradas:

- Sensor de presença

Estados:

- Luzes apagadas
- Luzes disponíveis

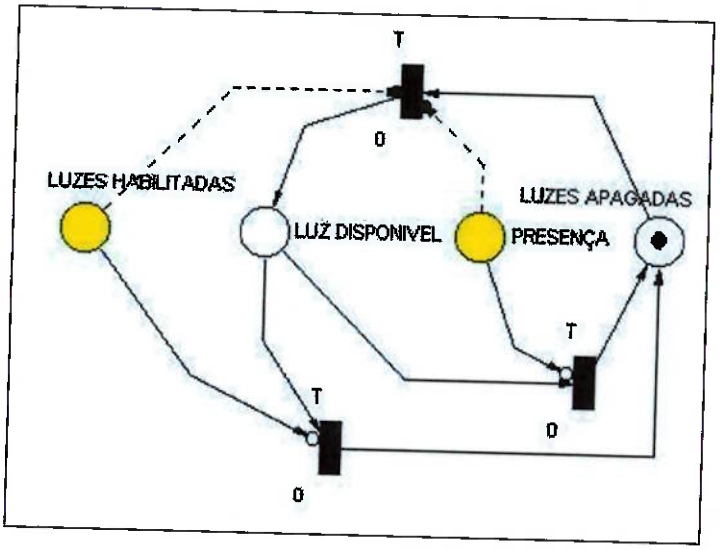

Figura 2-6: Rede de controle de Iluminação

2.3 *Sistema de Ventilação*

O sistema que será implementado é uma simplificação de um sistema real de conforto térmico em ambiente predial, que envolvem diversas salas e instalações mais complexas.

O sistema de ventilação permitirá que o usuário defina a temperatura que deseja para a sala através do sistema supervisor (entre 20°C e 26°C). Sempre que a sala estiver habilitada e seja detectada presença, o sistema manterá a temperatura em torno de 0,5°C da temperatura definida. Caso a sala esteja desocupada o ventilador funcionará com velocidade constante e baixa, para que economize energia e não cause desconforto excessivo quando a sala for ocupada.

A modelagem do sistema de ventilação foi feita em duas etapas para facilitar sua implementação.

Um sub-sistema verifica a necessidade de se manter, elevar ou baixar a temperatura. Esta informação é o dado de entrada para outro sub-sistema de controle do ventilador.

O ventilador possui dez diferentes níveis de velocidades e seu controle é dado pelo número de níveis ativados. Desta forma, enquanto a temperatura desejada for menor que a temperatura ambiente os níveis do ventilador serão ativados até atingir-se o equilíbrio. Na situação contrária os níveis serão gradativamente desativados.

2.3.1 Modelagem do Controle de temperatura

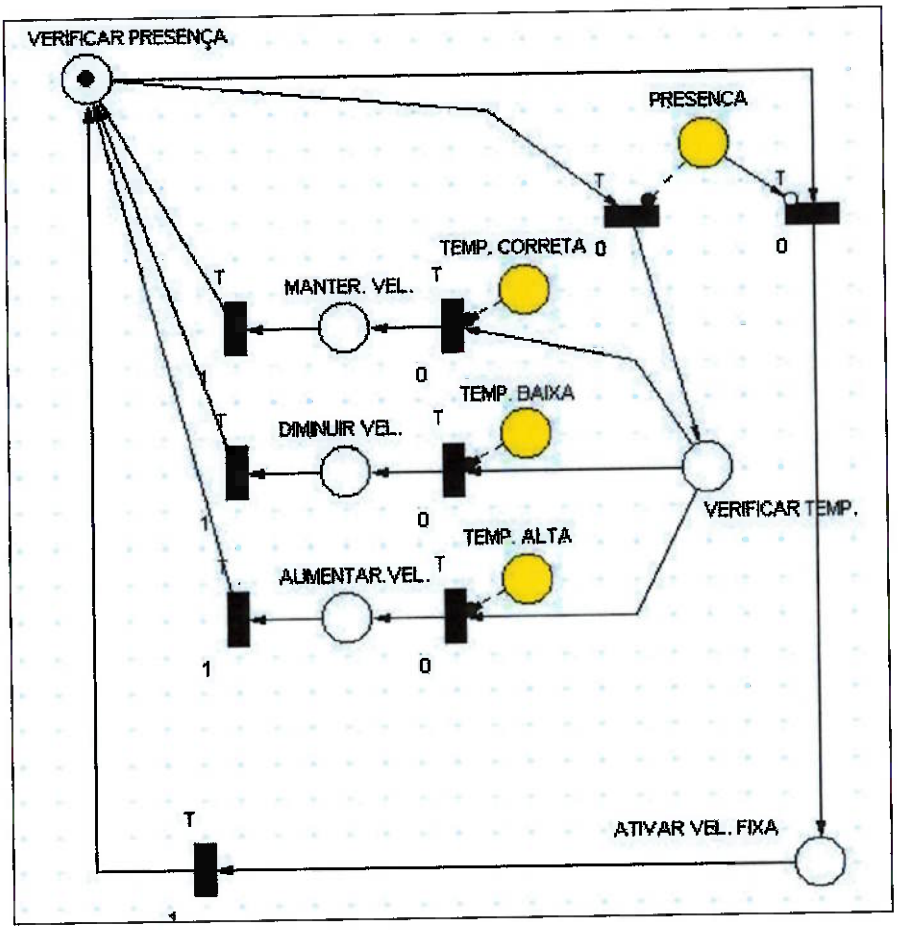

Figura 2-7: Rede de controle de Temperatura

Parâmetros:

- Temperatura desejada
- Temperatura atual

Entradas::

- Sensor de presença

Saídas:

- Atuação na velocidade do ventilador

2.3.2 Modelagem do ventilador

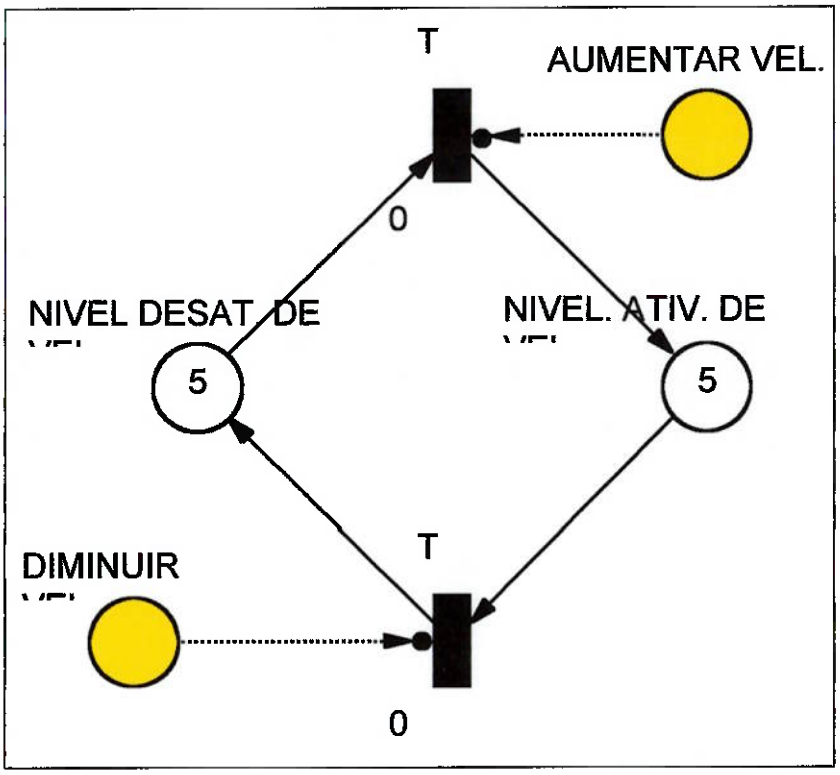

Figura 2-8: Rede de controle do Ventilador

Entradas:

- Aumentar velocidade
- Diminuir velocidade

Saída

- Níveis ativados do ventilador

2.4 Sistema de Segurança do Elevador

Quando o sistema de segurança for acionado, através do sinal de incêndio ou manualmente, verifica-se se o elevador está em movimento. Neste caso ele irá para o próximo andar e manterá a porta aberta até que não seja mais

detectado presença, quando a porta será fechada e travada, mantendo-se assim até que o sistema de segurança seja desativado.

Dessa forma pode-se garantir que em caso de incêndio, nenhum usuário utilizará os elevadores nem ficará preso dentro deles.

2.4.1 Modelagem do controle do elevador em caso de emergência

Parâmetro definido pelo supervisor:

- Condições normais

Entradas:

- Movimento do elevador
- Presença no elevador
- Sinal de incêndio

Estados:

- Emergência ativada
- Emergência desativada

Saídas:

- Controle do movimento do elevador
- Controle das portas do elevador

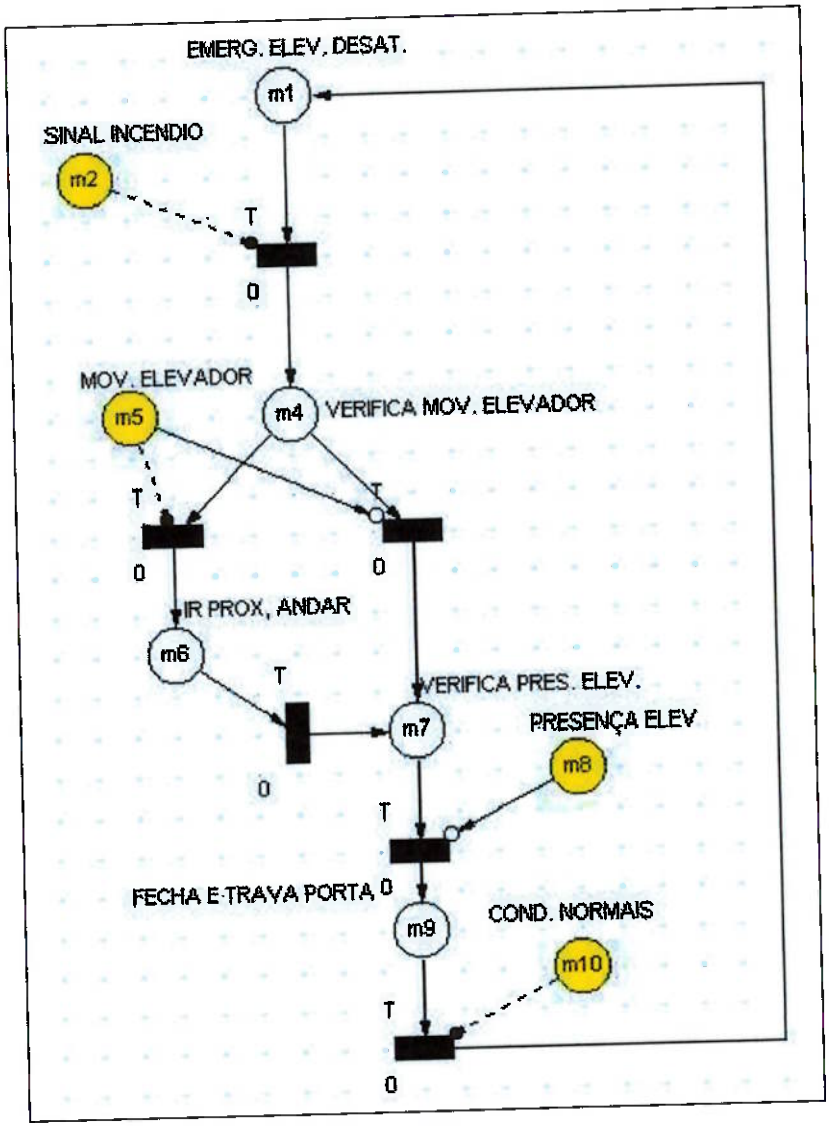

Figura 2-9: Rede de controle do elevador em emergência

3 Implementação

3.1 Controle das Salas

O primeiro e segundo andares serão controlados por quatro CLPs (PS4-102-MM1). Cada CLP controlará os sensores e atuadores de duas salas. Serão controlados os sensores de presença, fumaça, temperatura, um relé para acionamento do sistema de iluminação e um outro para o acionamento do sistema de ventilação. O controle de velocidade deste ventilador será feito por uma placa controladora conectada ao CLP.

3.2 Controle do Elevador

O elevador terá dentro da cabine um sensor de presença e um atuador na porta para sua abertura. Em cada andar será fixado um sensor de presença com a finalidade de detectar a posição do elevador.

Um motor de passo, controlado por pela placa da *National Instruments* que será detalhada em seguida, e acionado pelo programa *LabWindows* será responsável pela movimentação do elevador.

3.2.1 Elevador – Controle do movimento

O movimento do elevador será controlado através de um motor de passo conforme a figura a seguir:

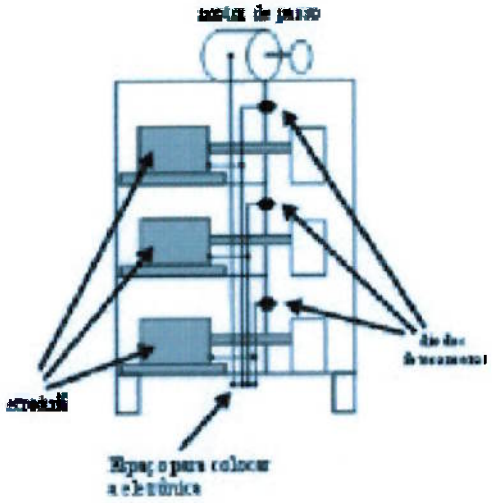

Figura 3-1: Esquema do elevador

A escolha do motor de passo deve as suas características de baixo custo e facilidade de controle. Sua precisão é limitada, porém suficiente para a aplicação.

O controle do motor será feito através do *LabWindows*, que a partir da compilação dos parâmetros monitorados enviará um sinal de controle ao motor. Os sinais de controle possíveis são: mover em sentido horário ou anti-horário, em *full* ou *half-step.*

O sinal proveniente do *LabWindows* irá para a placa da *National Instrument* e posteriormente para um circuito de potência e controle, que será apresentado a seguir, que efetivamente controlará o motor de passo.

O circuito de controle do motor de passo é composto essencialmente de dois CIs, o L297 e o L298. Onde o L297 gera o sinal de controle, enviando seqüencialmente os pulsos para as fases do motor e o L298 é utilizado como *driver.* A montagem do circuito é a seguinte:

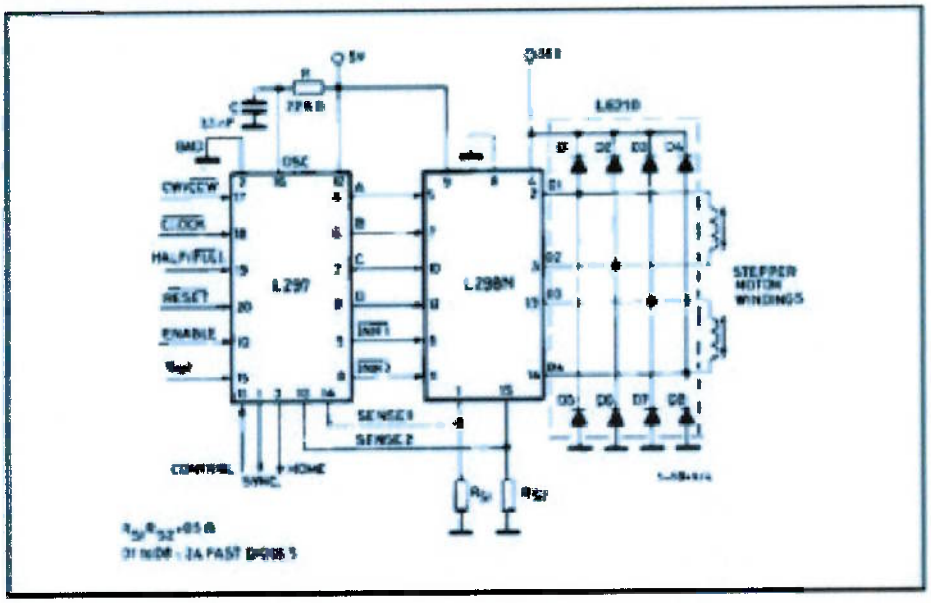

Figura 3-2: Circuito de controle do motor de passo

3.2.2 *Placa da National Instruments*

A placa da *National Instruments* utilizada para o controle do elevador é a PCI-6024E.

As principais características da placa são:

- 16 canais de entrada analógica
- 2 canais de saída analógica
- 8 portas de I/O digitais
- resolução máxima de 50ns

A imagem da placa segue abaixo:

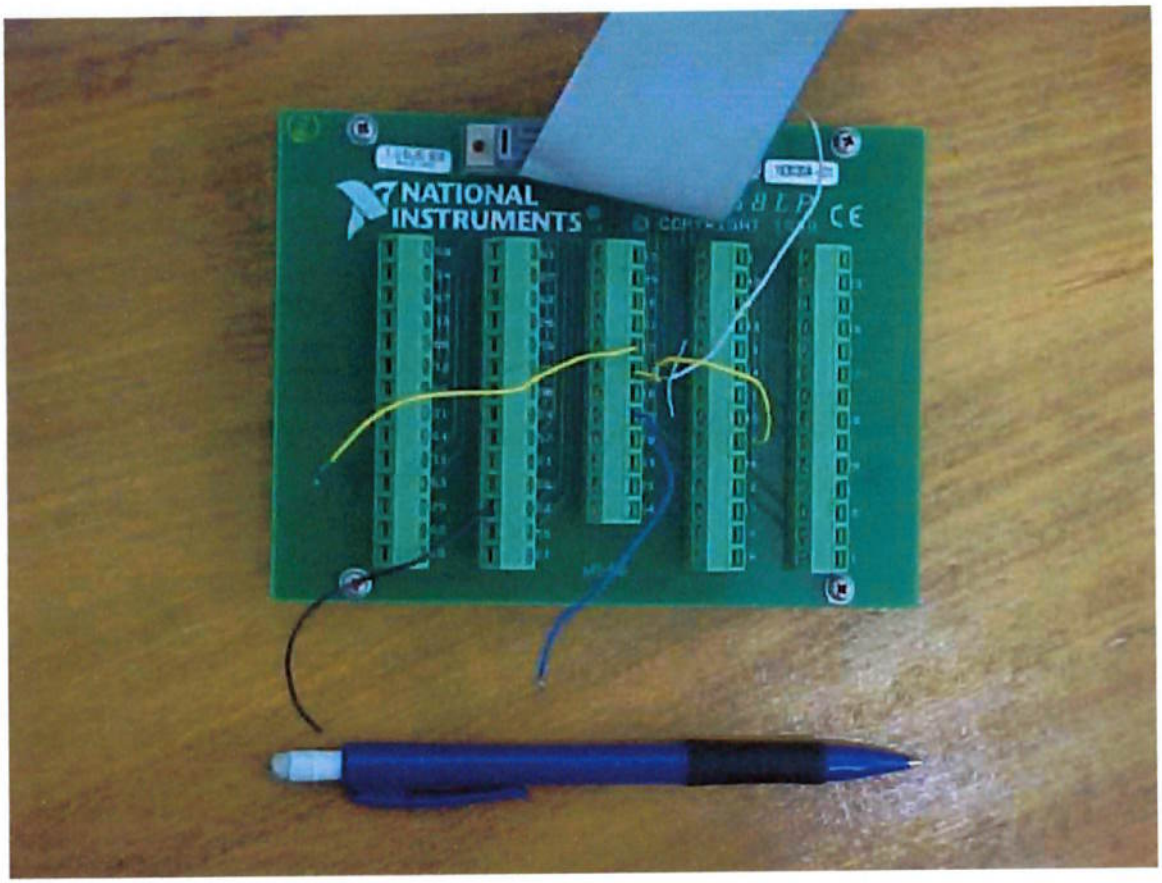

Figura 3-3: Placa para controle do elevador

Para o controle do elevador foram utilizadas seis portas digitais. (Portas 0 a 5). Correspondentes às conexões da placa números 52, 17, 49, 47, 51 e 19, respectivamente. Além da porta 18, onde deve ser ligado o terra.

As portas de 0 a 2 foram utilizadas para obter os sinais dos sensores de detecção do elevador; do térreo, primeiro e segundo andar, respectivamente.

A porta 3 foi utilizada para enviar ao *driver* do motor de passo o sentido de rotação. Horário (+5V) e anti-horário (0V).

A porta 4 foi utilizada para enviar o sinal de *enable* (+5V) e *disable* (0V) para o *driver*.

E a porta 5 foi utilizada para receber o sinal de presença no elevador, para detectar se ele esta ocupado ou não.

O sinal de *full* ou *half-step* não foi utilizado para o controle, pois neste trabalho não é feito o controle da velocidade do elevador. A entrada do *driver* foi colocada em 0V, o que corresponde ao movimento *full-step*, adotado como padrão neste trabalho.

3.3 Controle do Ventilador

O controle do ventilador foi feito através do uso de um circuito PWM – *Pulse Width Modulator*. Que faz uma modulação de voltagem através da variação da largura de pulso de voltagem.

Neste trabalho, utilizou-se o CI 3524 para realizar o controle. 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.

O sinal de entrada varia entre 0 e 100% de uma voltagem de referência, e a saída será 0 ou 100% da voltagem máxima. A velocidade do ventilador será proporcional ao tempo que o sinal de saída fica em 0 ou 100%. Isso é obtido através da comparação do sinal de entrada com a onda de forma triangular.
Assim, conforme o sinal de entrada aumenta, aumenta o tempo em que o sinal de saída fica em 100% e vice-versa.

O circuito de controle do PWM segue abaixo:

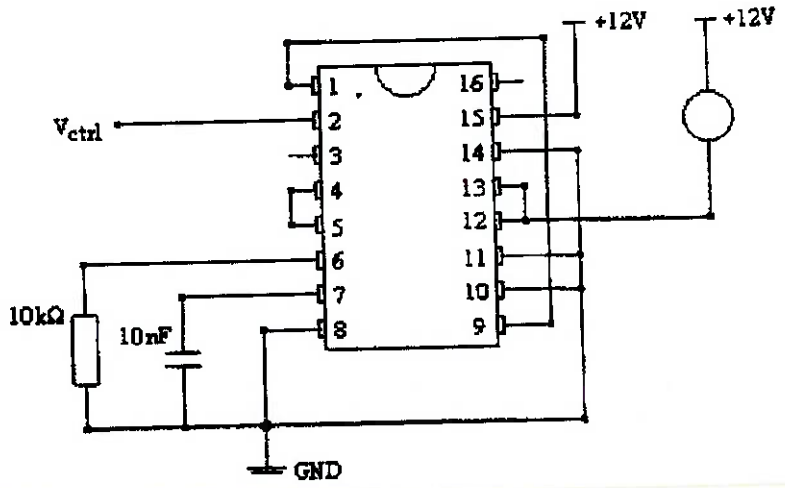

Figura 3-4: Circuito de controledo PWM

3.4 Conexão dos Sistemas

O programa *LabWindows* da *National Instruments* é responsável por coletar e processar os dados e atuar através de uma placa própria. A figura 3,1 representa o esquema do controle do sistema.

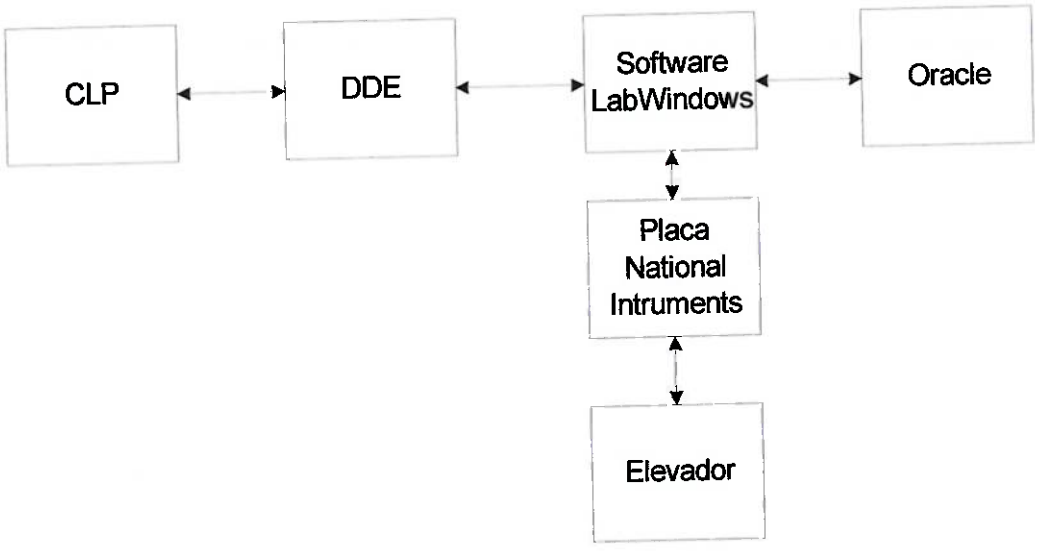

Figura 3-5: Esquema de controle do sistema

O controle do elevador é efetuado pela placa da *National Instruments*. Os dados serão trocados entre os CLPs e o programa *LabWindows* via *DDE*. As variáveis definidas pelo supervisor do sistema e os dados coletados pelo *LabWindows* são trocados com o programa supervisor através de um Banco de Dados *Oracle*.

4 Desenvolvimento do Software de Controle

4.1 CLP

O software implementado no CLP deve permitir o seu funcionamento independentemente do supervisor. Isto se deve a possíveis falhas na comunicação com o supervisor e a necessidade de ações quando for detectada alguma situação de emergência. Desta forma o CLP tem a função de ler os sinais dos sensores e de atuar nos ventiladores, luzes e sprinklers de acordo com dados padrões ou de acordo com dados enviados pelo supervisor.

Neste trabalho utilizou-se CLPs da Klöckner Moeller modelo PS4-201-MM1, para a aquisição de sinais e para atuar nas quatro salas da maquete projetada. Foram necessários quatro CLPs, um para cada sala, e por limitações do modelo de CLP a melhor solução encontrada foi utilizar um CLP como mestre (Master) e os outros 3 como escravos inteligentes (Slave). Nesta configuração todos os CLPs devem ter um programa de controle independente e suas variáveis devem ser trocadas com o CLP mestre para que este gerencie o fluxo de dados com o supervisor.

Portanto, na configuração de sistema escolhida todas as salas passam a ter um controle independente gerenciadas por uma das salas que faz a comunicação com o supervisor. A figura abaixo demonstra como foi montada a rede de CLPs.

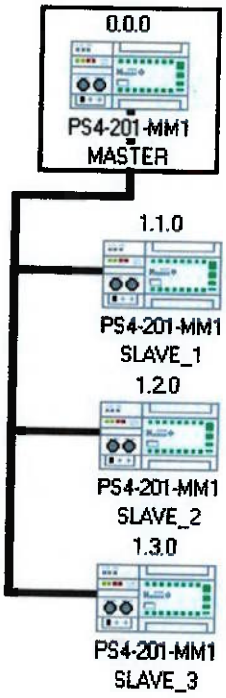

Figura 4-1: Configuração de conexão entre os CLPs

Na configuração demonstrada acima, as variáveis são trocadas por comandos de envio e recebimento de valores. Todos os sinais lidos nos CLPs escravos são enviados ao CLP mestre que fará a conexão com o supervisor. Da mesma forma as variáveis de intervenção do supervisor são enviados primeiramente ao CLP mestre que repassará ao CLP referente a sala que se deseja atuar.

A ligação dos CLPs implementada pode ser vista na foto abaixo:

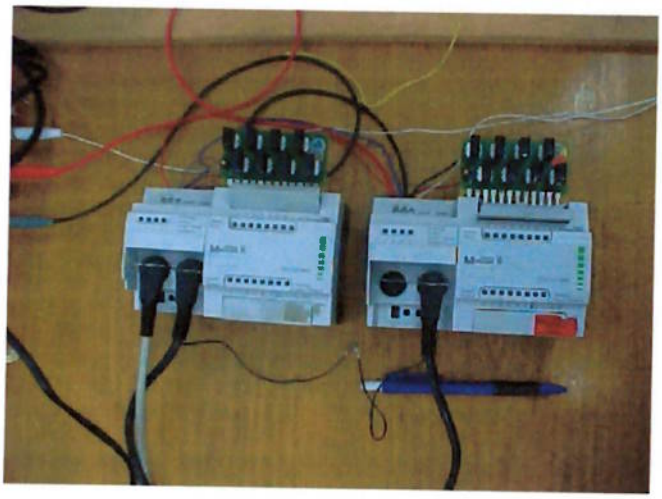

Figura 4-2: Foto da conexão entre os CLPs

A figura abaixo demonstra o fluxo de informações nos CLPs.

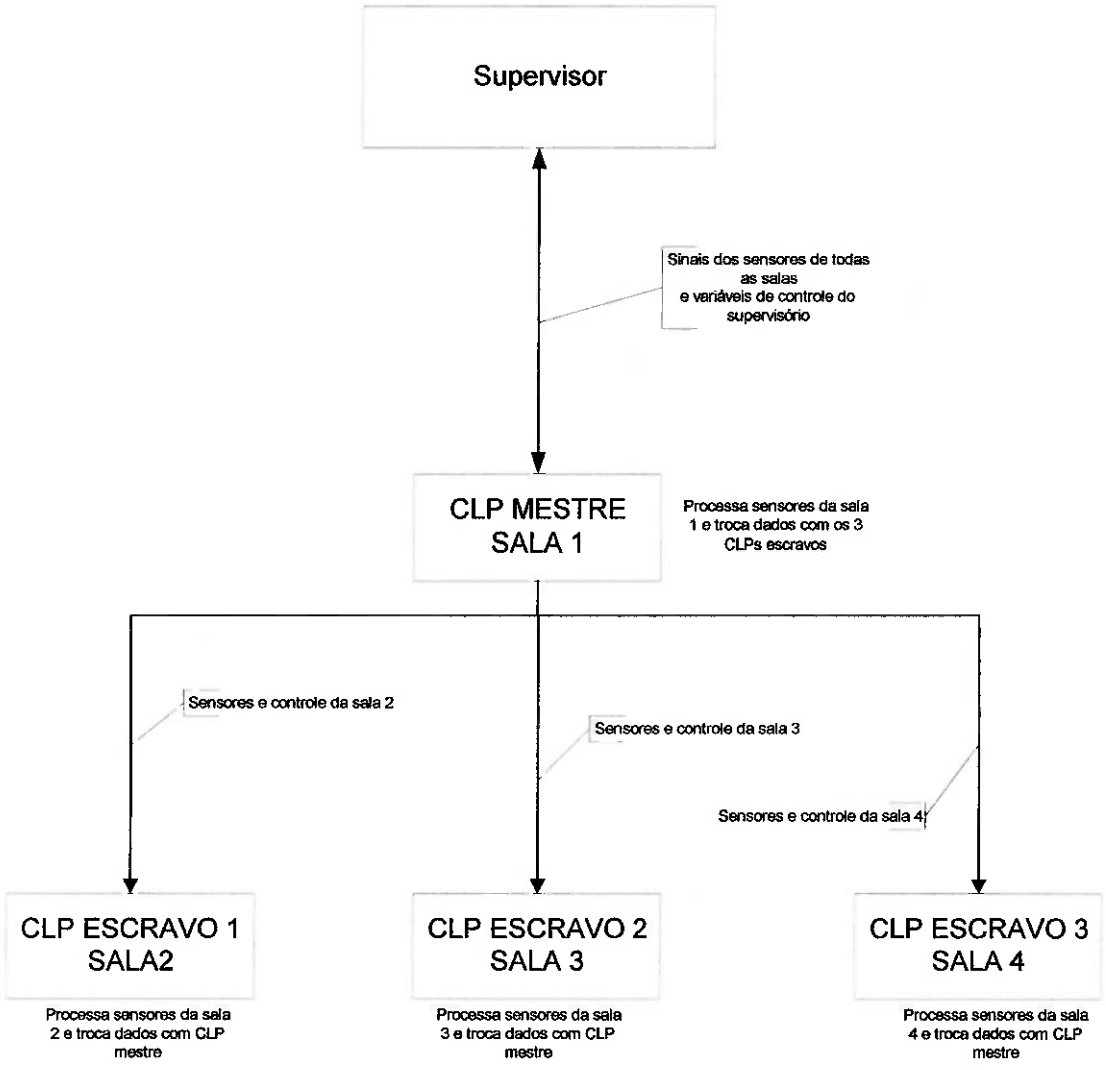

Figura 4-3: Troca de dados entre os CLPs

4.1.1 Módulo de Programa Principal do CLP Mestre

Como já foi descrito anteriormente o CLP principal tem a função de controlar não somente a sua sala, mas também controlar as transferência de dados entre o supervisor e os CLPs escravos. O diagrama abaixo demonstra o funcionamento do programa principal:

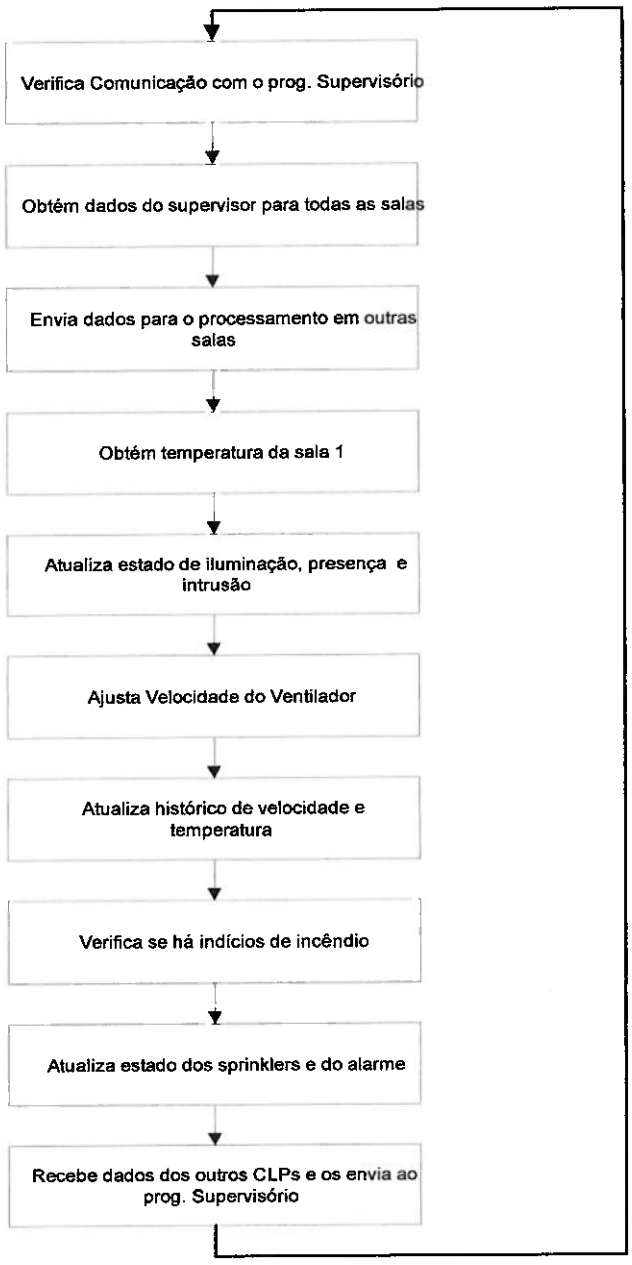

Figura 4-4: Fluxograma do programa principal do CLP mestre

4.1.2 Módulo de Programa Principal do CLP Escravo

A rotina principal do CLP escravo é semelhante a rotina principal para controle da sala 1 do CLP mestre. A principal diferença é que as variáveis de controle são recebidas e enviadas ao CLP mestre, que fará a interface com o supervisor.

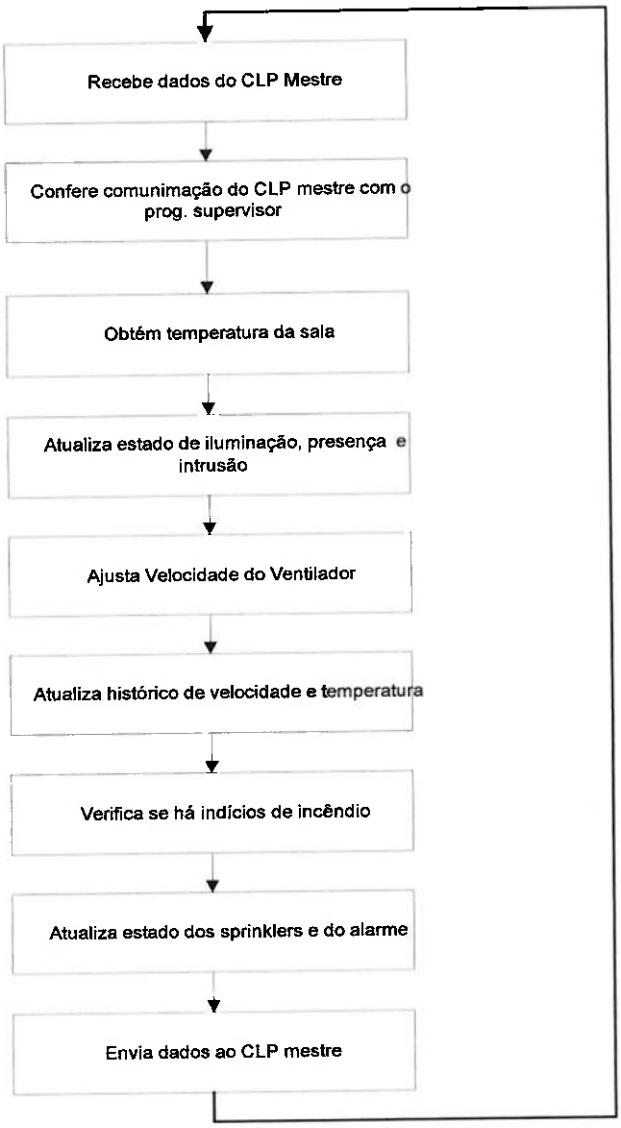

Figura 4-5: Fluxograma do programa principal do CLP escravo

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

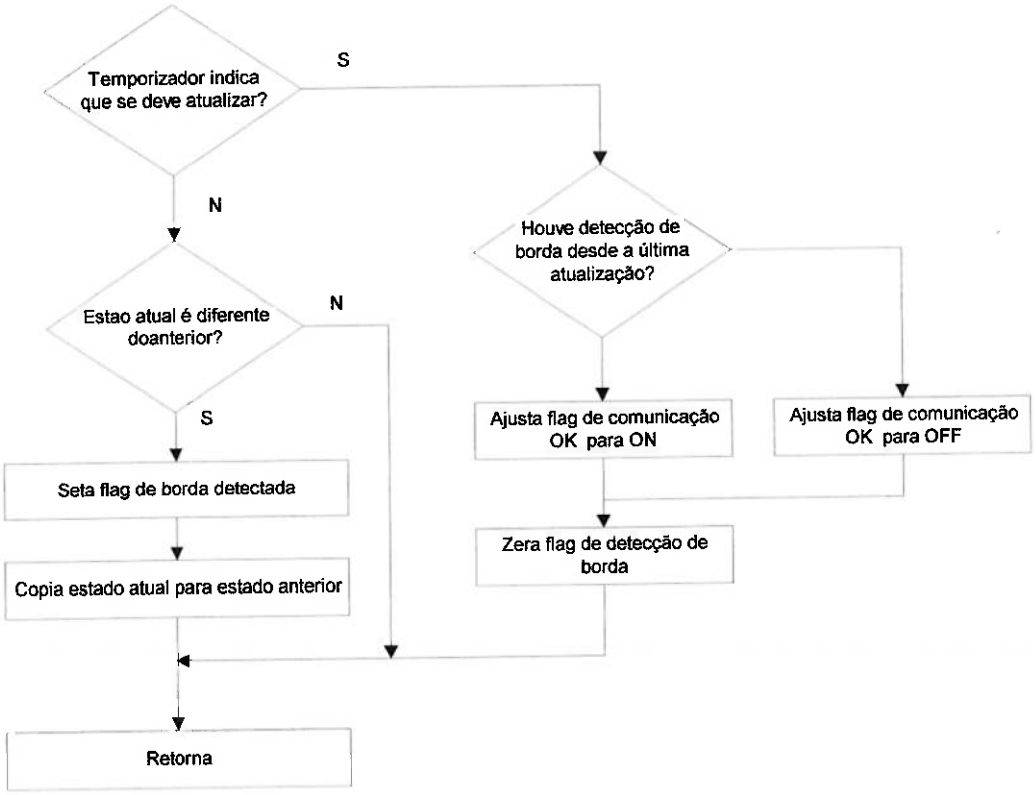

Figura 4-6: Fluxograma fa função FCNCOMUN

4.1.4 Função FCNLDTMP

Esta função permite que se faça a atualização das temperaturas máxima e mínima admissíveis na sala. Para isso, recebe como parâmetros a temperatura de ajuste enviada pelo supervisor, a temperatura ajustada no potenciômetro do próprio CLP e o estado da comunicação. Com estes dados a função escolhe se o ajuste a ser feito é em relação ao local ou ao supervisor.

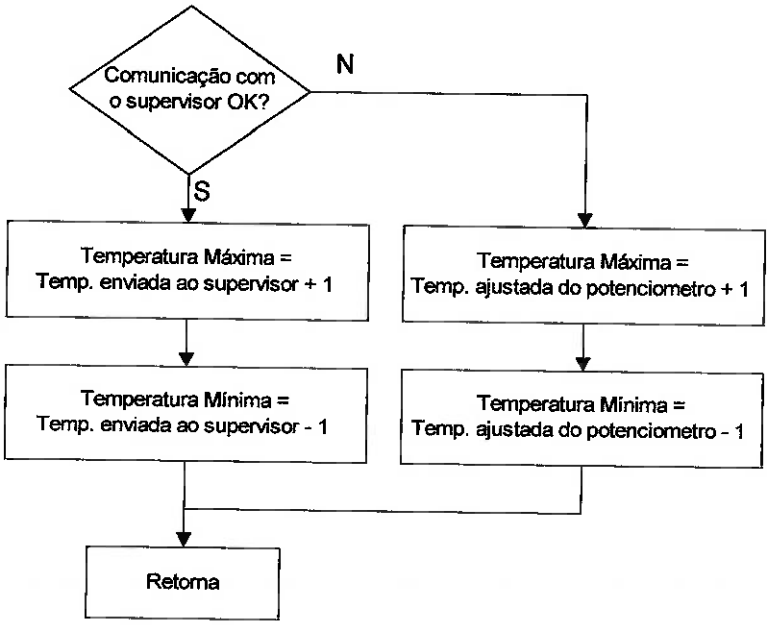

Figura 4-7: Fluxograma da função FCNLDTMP

4.1.5 Função FCN_INVA

Esta função recebe como entrada a indicação, a leitura de presença na sala e o estado da sala (habilitada/desabilitada) do supervisor. Com estes dados a função retorna o estado da sala (ocupação e se há intrusos) e habilita a acionamento das luzes.

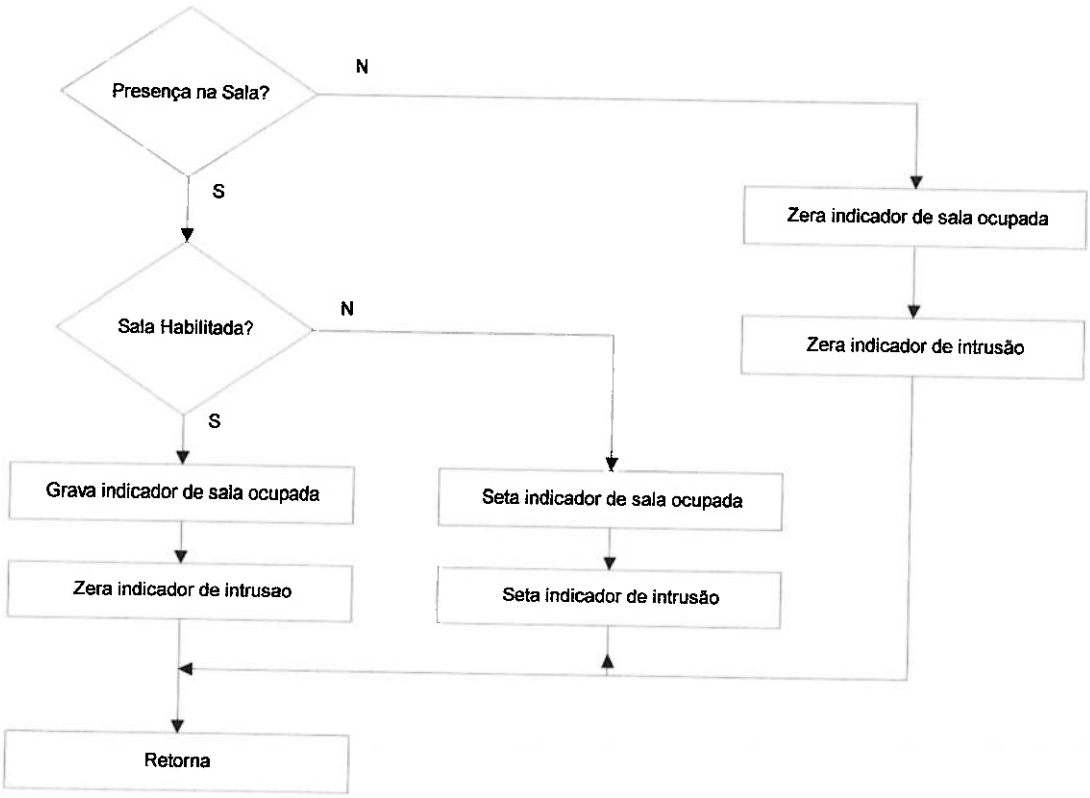

Figura 4-8: Fluxograma da função FCN_INVA

4.1.6 Função FCN_ALRM

Esta função recebe os seguintes sinais:

- indicador de possível 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 destes 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 levadas em conta. Isto pode ser visto no seu diagrama de funcionamento mostrado a seguir:

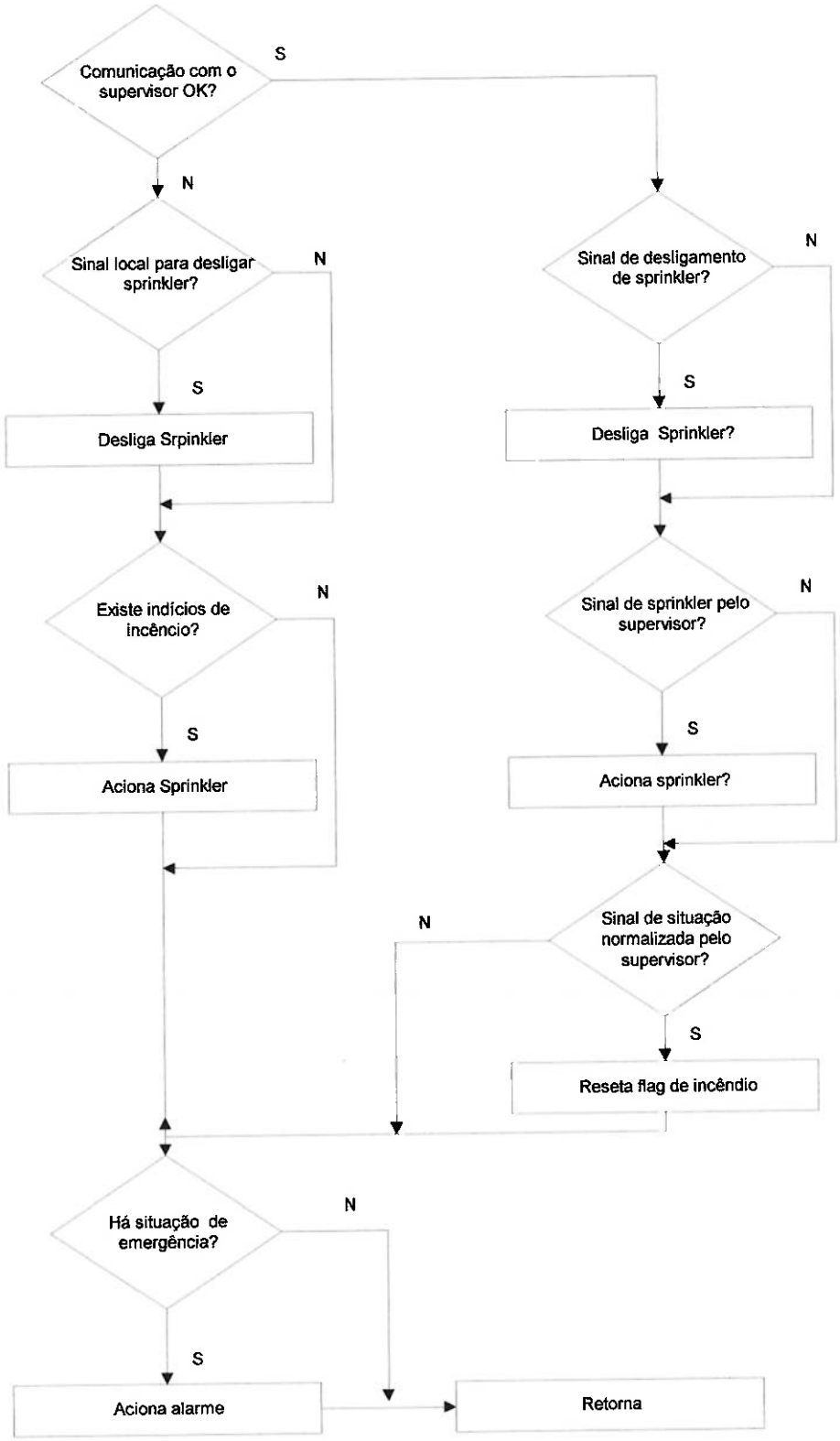

Figura 4-9: Fluxograma da função FCNALARM

4.1.7 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 cada vez que a velocidade seja alterada, aguarde-se um intervalo de 30s até que se possa ser novamente ajustada. Isso permite que a velocidade do ventilador não passe quase instantaneamente para 0 ou 100% conforma 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:

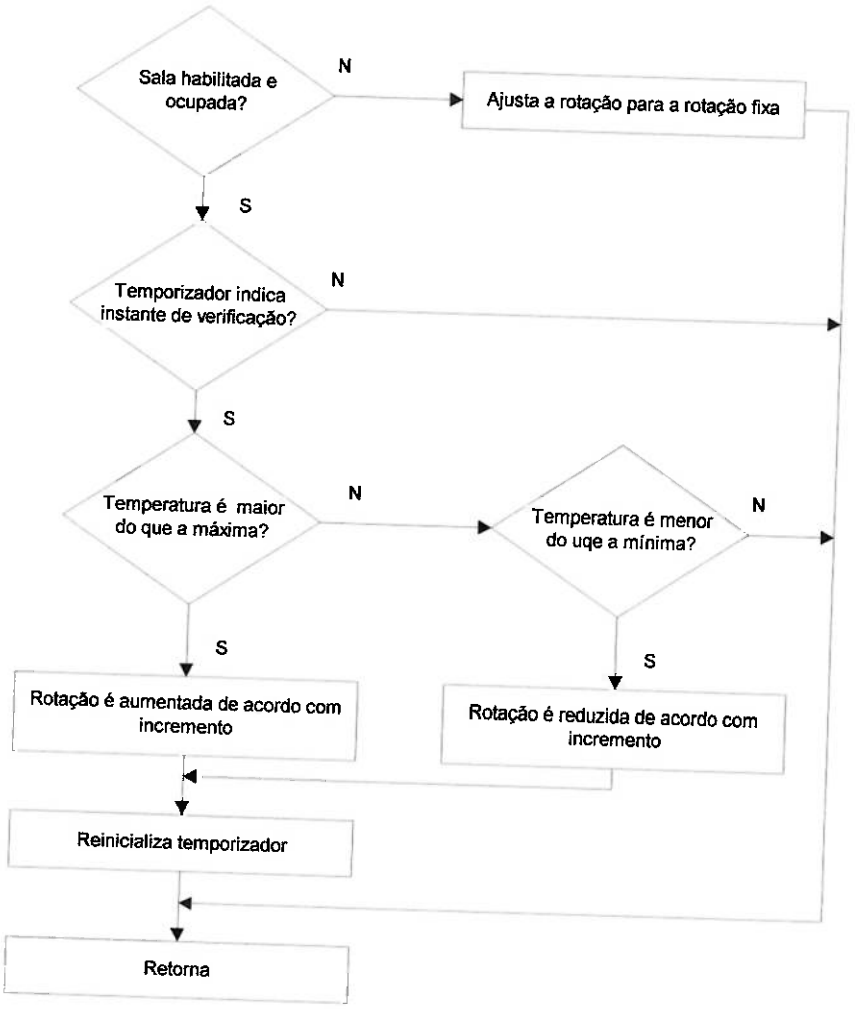

Figura 4-10: Fluxograma da função FCNVENTIL

4.2 Sistema Supervisor

4.2.1 Software

Como foi citado anteriormente o sistema supervisor foi desenvolvido no software *LabWindows/CVI 5.0*. A programação no *LabWindows* é feita em linguagem C e permite a criação de uma interface gráfica a partir de elementos pré-definidos como botões, Leds e displays.

4.2.2 Interface com o usuário

A partir da interface gráfica o usuário terá acesso ao controle dos parâmetros de temperatura, presença e do controle do elevador, assim como informações sobre o status do sistema; como presença de fumaça, fogo ou de invasão.

Os procedimentos de emergência, por questões de segurança, não poderão ser alterados pelo usuário comum. Apenas através do código fonte.

Procurou-se desenvolver uma interface simples e de fácil compreensão, de forma que qualquer usuário, mesmo sem treinamento, possa controlar o edifício sem colocar o mesmo em risco.

A interface é composta de 3 módulos básicos:

- Módulo do Elevador
- Módulo de Presença
- Módulo de Temperatura

Além de um módulo de teste, desenvolvido apenas para efeito de simulação, onde é possível simular diversos cenários. Este módulo não fará parte em um sistema real.

A interface desenvolvida segue abaixo:

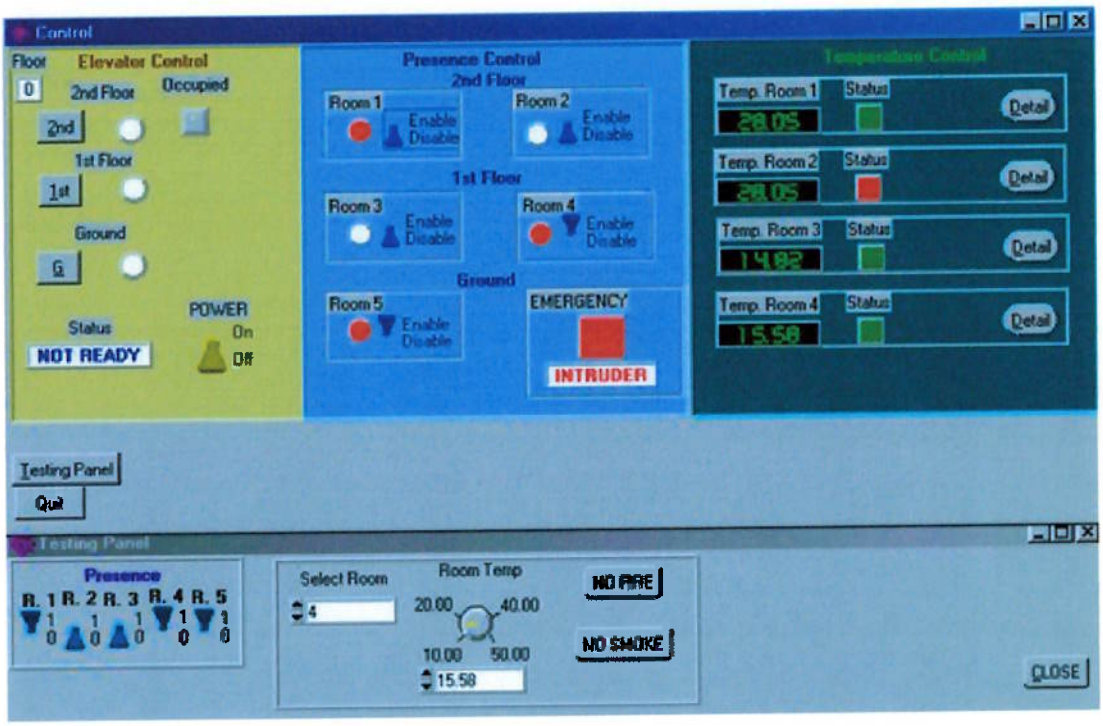

Figura 4-11: Painel de interface com o usuário

A seguir cada módulo será detalhado.

4.2.2.1 Módulo do Elevador

O módulo do elevador possui a interface apresentada abaixo:

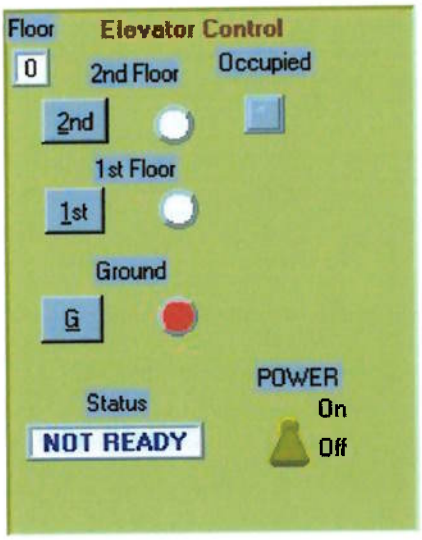

Figura 4-12: Módulo de Controle do Elevador

As funções presentes neste módulo são:

- *Power*

Este *switch* liga ou desliga o sistema do elevador. Quando desligado (off) todas as outras funcões do elevador são desabilitadas imediatamente, permanecendo assim até que seja novamente religado. Caso algum comando de movimentação seja acionado enquanto o *power*_estiver desligado, nada vai ocorrer, simulando a situação real em que um elevador está desligado e fora de serviço.

Com o power na posição *on* o elevador estará liberado para o uso e atenderá aos comandos de movimentação.

Ao se ativar o elevador uma rotina de inicialização é executada, visando verificar a posição inicial do elevador. Esta rotina funciona da seguinte maneira: primeiramente verifica-se o estado dos sensores de detecção do elevador, para detectar se este se encontra em um dos andares e não entre eles. No caso de um dos sensores estar ativo, o elevador estará pronto para uso, pois elevador já se encontra em um dos andares.

Caso todos os sensores estejam inativos, indicando que o elevador se encontra entre dois pisos, o motor de passo que controla o movimento do elevador será acionado no sentido horário (movimentando o elevador para baixo) até que um dos sensores detecte a presença do elevador. Nesse momento o movimento será interrompido e o elevador estará disponível para o uso.

- *Status*

O indicador de status indica o estado atual do elevador. Ele possui três indicações possíveis que são:

- Not Ready: indica que o elevador não esta disponível para uso, ou por estar desligado, ou por estar executando a rotina de inicialização.

- Ready: indica que o elevador esta disponível para uso

- Emergency: o indicador de emergência aparecerá em caso de detecção de fogo pelo sistema supervisor, neste momento o elevador entrará em uma rotina específica que será detalhada a seguir.

Procedimento de Emergência para o elevador

Caso seja detectado incêndio o movimento do elevador deverá ser interrompido, porém não deve prender pessoas dentro do mesmo. Para isso, os sensores de detecção do elevador são verificados, identificando se o elevador encontra-se em movimento ou parado.

Se estiver parado, o sensor de presença de pessoas dentro do elevador é verificado, se estiver inativo. Indicando que não há pessoas no elevador, as portas são fechadas e travadas. Se estiver ativo, indicando que há pessoas no elevador, as portas permanecerão abertas, até que não haja mais ninguém no elevador.

Caso o elevador esteja em movimento, ele continuará a se movimentar até o andar seguinte, quando será executado o procedimento descrito acima para a condição de elevador parado.

O status permanecerá na condição de emergência até que o procedimento de incêndio seja desabilitado. Neste momento o elevador passará para a condição desligada (*not ready*) e terá que ser religado para voltar ao funcionamento normal.

- *Botões de Comando*

Os três botões de comando (*G, 1st, 2nd*) movimentarão o elevador para os andares térreo, primeiro e segundo, respectivamente, desde que o elevador esteja ligado. Caso contrário, não surtirão nenhum efeito.

- *Leds*

Os *Leds* estão ligados aos sensores de detecção do elevador e indicam o andar em que o elevador se encontra.

- *Floor*

O indicador *floor* indica numericamente o andar atual do elevador.

- *Occupation*

O *Led* de ocupação está ligado ao sensor de presença no elevador, indicando se o mesmo esta ocupado ou não.

4.2.2.2 Módulo de Presença

Abaixo segue o painel do módulo de presença:

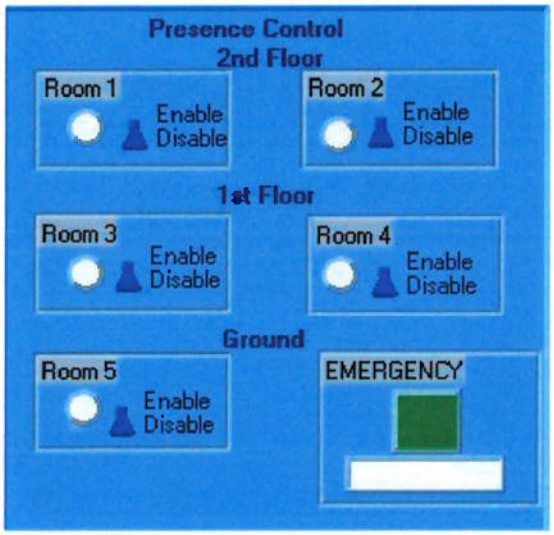

Figura 4-13: Módulo de Presença

O módulo de presença é subdividido em cinco partes análogas, uma para cada sala. Além do indicador de invasão, comum para as cinco salas.

É através deste módulo que o usuário define a política de ocupação das salas. Esta pode estar habilitada ou desabilitada para o uso, através dos *switchs* nas posições *enable* ou *disable*, respectivamente.

Existe ainda um *Led* para cada sala, que se acenderá em caso de detecção de presença na mesma.

Na situação em que a sala está habilitada para o uso, poderá ser ocupada normalmente e as luzes estarão disponíveis para serem acesas. Caso não esteja habilitada, as luzes não poderão se acesas e se for detectada presença na sala o indicador de invasão será acionado, conforme a figura a seguir:

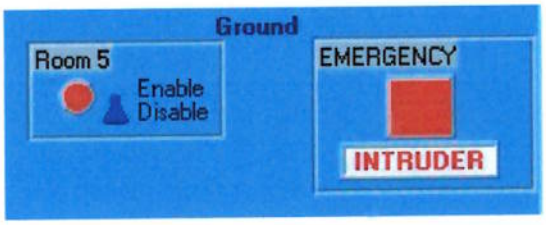

Figura 4-14: Indicador de Invasão

Este controle de presença e de disponibilidade da iluminação nas salas permite o uso mais racional da energia. Principalmente condiderando-se locais com um grande número de ambientes. Pois através da desabilitação das salas em horários de desuso, como madrugada por exemplo, garante-se de uma forma simples e eficiente que nenhuma luz ficará acesa desnecessariamente.

Além disso, o controle de invasão está centralizado em um mesmo ponto para todas as salas, possibilitando a tomada de atitudes cabíveis de forma mais rápida e sistemática.

4.2.2.3 Módulo de controle de temperatura

O módulo de controle de temperatura está disponível para as duas salas do primeiro e do segundo andar. O painel de temperatura segue abaixo:

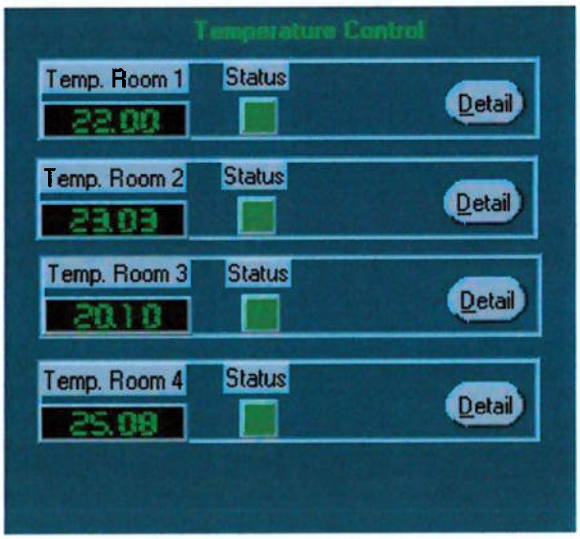

Figura 4-15: Módulo de Controle de Temperatura

Este painel indica a temperatura atual em graus Celsius de cada sala, assim como o status da mesma. O *Led* de status se acenderá caso seja detectado fumaça ou fogo na sala. Os sinais de fumaça ou fogo assim como o valor da temperatura atual, são provenientes dos CLPs, que realizam um monitoramento constante dos ambientes.

Para cada sala existe o botão de *Detail* que abre uma nova janela com informações detalhadas sobre aquela sala, conforme figura a seguir:

Dessa forma o sistema supervisor pode ser testado sem a necessidade da comunicação com os CLPs.

O painel de teste segue abaixo:

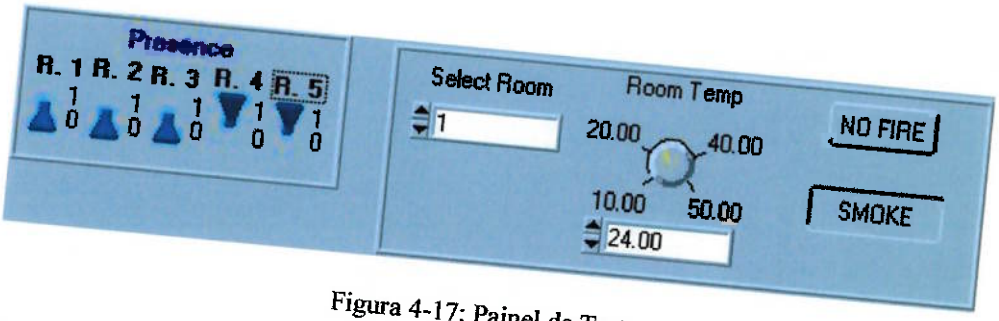

Figura 4-17: Painel de Teste

Do lado esquerdo tem-se os controles para simular os sinais provenientes dos sensores de presença. Caso o sensor esteja ativo, detectando presença, o *switch* deve estar na posição 1 e em 0 caso contrário.

Do lado direito tem-se os controles referentes à temperatura:

- *Select Room:* seleciona-se a sala para qual os valores serão atribuídos.
- *Room Temp:* regula-se a temperatura da sala selecionada
- *Fire / Smoke:* são botões de duas posições que simulam o sinal normal ou de fogo e fumaça para cada sala.

4.2.3 Função FCNARRAY

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

4.2.4 Função SPDCONVERT e TEMPCONVERT

A função SPDCONVERT realiza a conversão de um número que indica a velocidade percentual 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 graus Celsius em um número de 11 bits correspondente a entradaanalógica do CLP. Como retorno ela fornece a temperatura em graus Celsius correspondente a essa entrada admitindo que o limite superior edo range de temperatura seja correspondente a 10V e o inferior a 0V.

5 Conclusões Considerações para o Futuro

A partir do projeto desenvolvido neste texto, pode-se citar como próximos passos para melhorar e continuar a desenvolver sistemas de automação predial os seguintes itens:

- Integração do CLP com o LabWindows através de uma interface DDE;
- Definir o método de cabeamento dos sensores pela estrutura da maquete a fim de otimizar o espaço disponível;
- Integração do sistema de automação predial com um supervisor via internet utilizando-se da mesma base de dados para evitar duplicidade;
- Implementação de base de dados Oracle integrada ao LabWindows para armazenamento de dados;
- Análise dos dados coletados e definir uma estratégia para otimização das salas pelo supervisor;
- Implementação da câmera no térreo e sua integração à base de dados Oracle;

Este projeto permitiu o desenvolvimento de técnicas de gerenciamento de dados e sinais adquiridos para melhorar os sistemas de automação de dados já existentes. O esforço para que se mantivesse a coerência nos dados lidos em diferentes partes do projeto foi o principal objetivo e direcionamento do projeto. Também foi possível desenvolver técnicas de estruturação e programação de CLP e sistemas de aquisição e análise de sinais.

6 Bibliografia

[1] Del Foyo, P.G. y Silva, J.R. *Sistema Supervisório Descentralizado Baseado em Íntegrons*,, Universidade de São Paulo, Escola Politécnica.

[2] Silva, J.R.(1998). *Interactive Design of Integrated Systems*, BASYS 98, Praga, Tchec Republic, 1998.

[3] Silva, José Reinaldo, e Ramos, Roberto L. C.Barroso Ramos. *Controle Integrado: Aplicação em Sistemas Prediais.* 3° SBAI, Vitória, 1997

[4] Silva, José Reinaldo, Miyagi, Paulo e. *PFS/MFG: A High Level Net for the Modeling of Discrete Manufacturing System.* in Balanced Automation Systems, IEEE/ECLA/IFIP Proceedings, vol.I – Architectures and desingn methods, 1995.

[5] 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 Chanllenges for anthropocentric manufacturing, 1995

[6] Del Foyo, P.G., *GHENeSys: Uma Rede Estendida Orientada a Objetos para Projetos de Sistemas Discretos.* Tese de mestrado, Prof. José Reinaldo Silva, orientador. 2001

APENDICE I. LISTAGEM DAS FUNÇÕES DE CONTROLE

Apêndice I.1 Programa Principal do CLP Mestre

```
VAR
        (*Comunicacao com o 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;

        (*Variaveis enviadas ao Supervisorio da Sala 1*)
        Incendio_1          AT %RD1.1.0.2.0 : BOOL :=0;
        Intrusao_1          AT %RD1.1.0.4.0 : BOOL;
        SalaOcupada_1               AT %RD1.1.0.6.0 : BOOL;
        TempAtual_1         AT %RDB1.1.0.8  : BYTE;
        VelAtual_1          AT %RDB1.1.0.10  : BYTE;
        FumacaSala_1        AT %RDB1.1.0.12  : BYTE;
        AckSitNormal_1              AT %RD1.1.0.14.0 : 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;
        SituacaoNormal      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;

        (*Entradas provenientes da Sala 1*)
        SalaHabilitada_1    AT %SD1.1.0.30.0  : BOOL;
        Emergencia_1        AT %SD1.1.0.16.0  : BOOL;
        UserTemp_1          AT %SDB1.1.0.18   : BYTE;
        SinalAtual_1        AT %SDB1.1.0.20  : BYTE;
```

```
SituacaoNormal_1    AT %SD1.1.0.22.0 : BOOL;
SupervIncendio_1    AT %SD1.1.0.24.0 : BOOL;
IncrementoPerc_1    AT %SDB1.1.0.26  : BYTE := 5;
VelocFixa_1         AT %SDB1.1.0.28  : BYTE := 20;


MaxTemp : UINT;
MinTemp : UINT;

(*Entradas da Maquete*)
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 %Q0.0.0.0.0 : 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   :     FCN_ALRM;
        Ctrl_Acesso   :     FCN_INVA;
        Ctrl_Temp     :     FCNVENTIL;
        Ctrl_Array    :     FCNARRAY;

        (*VerificaComunic :  FCNCOMUN;*)
        LoadInputTemp  :    FCNLDTMP;

        (*Funcoes de Conversao*)
        SPEEDCONVERT   :        SPDCONV;
        CURRENTTEMP    :        TEMPCONV;
END_VAR

(*Considera comunicacao com o supervisorio OK*)
LD    1
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

(*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 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
EQ   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_SituacaoNormal   := SituacaoNormal,
```

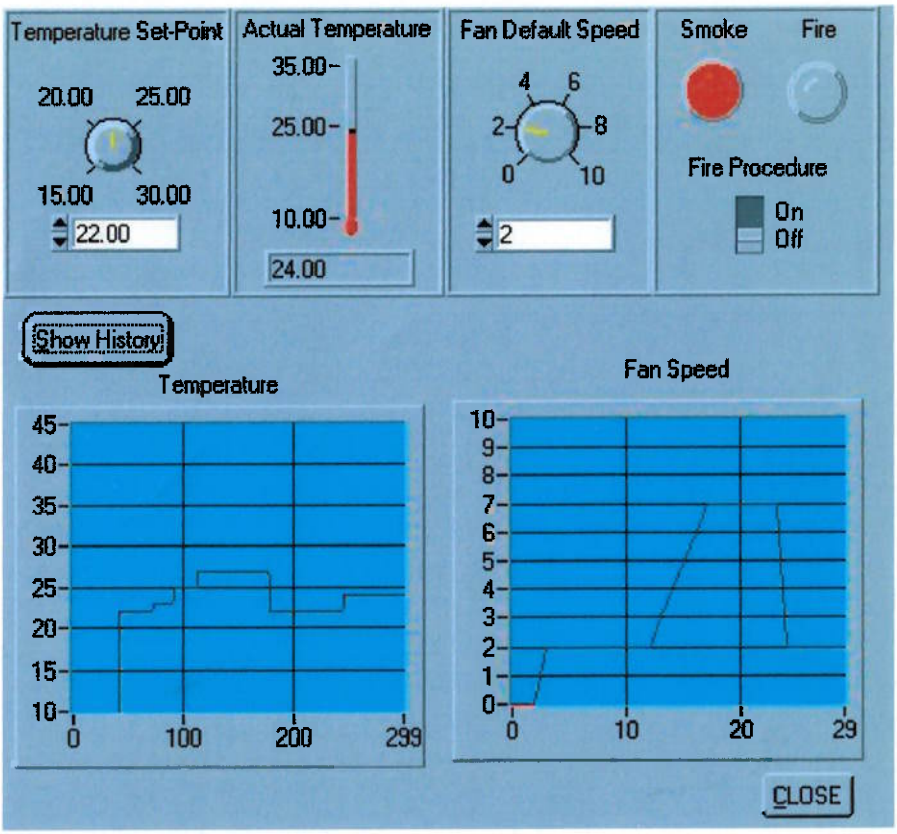

Figura 4-16: Painel com informações detalhadas referente à temperatura

Este painel está subdividido em:

- *Temperature Set-Point*

Onde o usuário seleciona a temperatura desejada para a sala na situação em que a estiver habilitada e com presença.

- *Actual Temperature*

Onde é indicada a temperatura atual da sala.

- *Fan Default Speed*

Esta opção permite o usuário definir uma velocidade padrão de funcionamento para o sistema de ventilação da sala na situação em que a estiver desocupada. O sistema de ventilação possui dez níveis de operação e caso o usuário deseje manter a temperatura da sala mais baixa, independente da presença, deve regular esta opção para um valor maior. Caso deseje aumentar a economia de energia, deve regular esta opção para um

valor mínimo; havendo neste caso o inconveniente de o sistema requerer um tempo maior para atingir a temperatura de *set-point* quando a sala for ocupada. (Considerando a temperatura de *set-point* menor que a temperatura atual).

Na situação em que a sala estiver habilitada e ocupada o sistema de ventilação buscará a velocidade ideal para atender à temperatura de *set-point* através da rotina já apresentada.

- *Fire / Smoke*

Os indicadores de fumaça e fogo serão acionados, caso o sistema supervisor receba esta informação do CLP. Na figura apresentada, o sensor de fumaça está ativo e o de incêndio inativo. Indicando que foi detectada a presença de fumaça na sala, mas devido ao histórico e à temperatura atual ainda não foi caracterizada uma situação de incêndio.

Caso o incêndio seja detectado o indicador se acenderá e o *Fire Procedure* passará para a posição *on.* Dando início ao procedimento de emergência previamente apresentado. Porém, se o usuário definir que a situação não é de incêndio, apesar da indicação do CLP. Basta ele desligar o *Fire Procedure*, que o procedimento de emergência será interrompido. O usuário possui também a opção de acionar o procedimento de emergência a qualquer momento, independente da indicação do CLP.

- *Show History*

Se o botão de *Show History* for acionado, será plotado nos gráficos os históricos de temperatura e da velocidade do ventilador dos últimos 5 minutos. Isso permite o usuário monitorar a temperatura e verificar a eficiência do sistema de ventilação.

4.2.2.4 Módulo de Teste

Como citado anteriormente foi criado um painel de teste, para a realização de testes e simulações de cenários. Este módulo simula todas as entradas provenientes do CLP.

```
                In_ComunicOK          := ComunicOk,
                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


LD    Fumaca
ST    FumacaSala.0

SendComunicOK:
LD    ComunicOK
ST    Q07
```

Apêndice I.2 Programa Principal do CLP Escravo

```
VAR
        (*Comunicacao com o Supervisorio*)
        ComunicOK : BOOL :=0;
        UltSinal : Byte :=0;
        SinalOK : BOOL :=0;

        (*Variaveis enviadas ao Supervisorio*)
        Incendio            AT %SD0.0.0.2.0 : BOOL :=0;
        Intrusao            AT %SD0.0.0.4.0 : BOOL;
        SalaOcupada         AT %SD0.0.0.6.0 : BOOL;
        TempAtual           AT %SDB0.0.0.8  : BYTE;
        VelAtual            AT %SDB0.0.0.10  : BYTE;
        FumacaSala          AT %SDB0.0.0.12  : BYTE;
        AckSitNormal        AT %SD0.0.0.14.0 : BOOL:=0;

        (*Entradas provenientes do Supervisorio*)
        SalaHabilitada      AT %RD0.0.0.30.0  : BOOL;
        Emergencia          AT %RDM0.0.0.16.0  : BOOL;
        UserTemp            AT %RDB0.0.0.18   : BYTE;
        SinalAtual          AT %RDB0.0.0.20  : BYTE;
        SituacaoNormal      AT %RD0.0.0.22.0 : BOOL;
        SupervIncendio      AT %RD0.0.0.24.0 : BOOL;
        IncrementoPerc      AT %RDB0.0.0.26  : BYTE := 5;
        VelocFixa           AT %RDB0.0.028   : BYTE := 20;

        MaxTemp : UINT;
        MinTemp : UINT;

        (*Entradas da Maquete*)
        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 %Q0.0.0.0.0 : 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   :     FCN_ALRM;
    Ctrl_Acesso   :     FCN_INVA;
    Ctrl_Temp     :     FCNVENTIL;
    Ctrl_Array    :     FCNARRAY;

    (*VerificaComunic :  FCNCOMUN;*)
    LoadInputTemp  :    FCNLDTMP;

    (*Funcoes de Conversao*)
    SPEEDCONVERT  :     SPDCONV;
    CURRENTTEMP   :     TEMPCONV;
END_VAR

(*Considera comunicacao com o supervisorio OK*)
LD    1
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

(*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 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
EQ    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_SituacaoNormal     := SituacaoNormal,
                   In_ComunicOK                := ComunicOk,
                   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


LD    Fumaca
ST    FumacaSala.0

SendComunicOK:
LD    ComunicOK
ST    Q07
```

Apêndice I.3 Função FCN_ALRM

```
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 com o supervisorio OK*)

LD      In_ComunicOK
JMPCN       ComunicLost

LD      In_SituacaoNormal
ST      Ext_AckSitNormal

LD      In_SituacaoNormal
ANDNIn_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
```

Apêndice I.4 Função FCNARRAY

```
VAR_INPUT
        In_Temperatura : UINT ;
        In_VelocPerc : INT ;
END_VAR
VAR_OUTPUT
        Out_TempCrescente : BOOL ;
        Out_VelCrescente : BOOL ;
END_VAR
VAR
        TrigGetTemp : BOOL:=0 ;
        DTEmpGetTemp : TIME := T#1s ;
        TimerGetTemp : TP ;
        Array_Temp : ARRAY [1..20] OF UINT;
        Array_Vel : ARRAY [1..20] OF INT;
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]
GT    Array_Vel[1]
JMPC  CompVel2
LD    0
ST    Out_VelCrescente

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

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

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

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

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

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

```
      CompVel8:
LD    Array_Vel[8]
GT    Array_Vel[8]
JMPC  CompVel9
LD    0
ST    Out_VelCrescente


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

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

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

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

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

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

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

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

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

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

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

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

EndGetArrays:
```

Apêndice I.5 Função FCN_INVA

```
VAR_INPUT
        In_Presenca : BOOL ;
        In_SalaHabilitada : BOOL ;
END_VAR
VAR_OUTPUT
        Out_SalaOcupada : BOOL ;
        Out_Intrusao : BOOL ;
        Out_Luzes : BOOL ;
END_VAR
(*Algoritmo de controle de acesso a sala*)

LD      In_Presenca
JMPC  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:
```

Apêndice I.6 Função 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 do Superv*)
BYTE_TO_UINT
ADD     1
ST      Out_MaxTemp
SUB     2
ST      Out_MinTemp
JMP     EndCarregaUserTemp

NoComunic:

CAL     POTENCTEMP(TEMPWORD:=In_PotencTemp,
                   TEMPMAX:=26,
                   TEMPMIN:=20)
LD      POTENCTEMP.TEMPGRAUS
BYTE_TO_UINT
ST      TempPotenc (*Temp.do potenciometro*)
ADD     1
ST      Out_MaxTemp
SUB     2
ST      Out_MinTemp
EndCarregaUserTemp:
```

Apêndice I.7 Função 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      Incremento

(*Controle do Ventilador*)
ContrVentilador:
CAL     Timer(IN:=CounterFlag,
            PT:=Delay)
LD      Timer.Q
EQ      1
R       CounterFlag
JMPCN       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:
```

Apêndice I.8 Função 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
```

Apêndice I.9 Função TMPCONV

```
VAR_INPUT
        TEMPWORD : WORD ;
        TEMPMIN : UINT ;
        TEMPMAX : UINT ;
END_VAR
VAR_OUTPUT
        TEMPGRAUS : BYTE;
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
```

APENDICE II. LISTAGEM DO PROGRAMA DO SISTEMA SUPERVISOR

Apêndice II.1 Listagem do programa em Superviosor.C

```
#include <ansi_c.h>

#include <Dataacq.h>
#include <cvirte.h>          /* Needed if linking in external compiler; harmless
otherwise */
#include <userint.h>
#include "Supervisor.h"


#define enable 1
#define disable 0
#define CW 1
#define CCW 0
#define ON 1
#define OFF 0
#define vetor_size 300
#define vetor_fan_size 30

//------------
// Prototypes
//------------

static int control;
static int testing;
static int tempRoom1;
static int tempRoom2;
static int tempRoom3;
static int tempRoom4;
static int initialize(void);
static int check_ground(void);
static int check_first(void);
static int check_second(void);
static int check_presence(void);
static int update_leds_presence(void);
static int display_emergency(void);
static int check_permission(void);
static int check_temp(int);
static int check_fan(int);
static int selected_room(void);
```

```
    static int update_fan_matrix(int,int,int);
static int check_smoke_fire(void);
static int elevator_status_display(int);
static int fire_actions(void);



//variaveis globais
int daqErr;             /* status da linha digital*/
short   instate0,
                instate1,
                instate2,
                instate3,
                instate4,
                instate5,
                instate6,
                instate7,  /*Entradas Digitais*/
                outstate0,
                outstate1,
                outstate2,
                outstate3,
                outstate4,
                outstate5,
                outstate6,
                outstate7,                          /* Saidas Digitais*/
                presenca_elev;                      /*sensor de presenca no elev*/
int             actual_floor,                       /*posicao atual do elevador*/
                sensor0,sensor1, sensor2,      /*variaveis dos sensores de posicao */
                powerstate,                    /*status do elevador, habilitado ou não*/
                occupation[6],                 /*vetor de presenca nas salas*/
                permission[6],          /*vetor de permissão de presença */
                fan_historic[5][vetor_fan_size],
                fanspeed[5],                        /*armazena vel. atual do vent.*/
                smoke[5],               /*armazena status de fumaça das salas*/
                fire[5],          /*armazena status de incêndio das salas*/
                i=1,              /*variavel aux. utilizada p/ atualizar vetor de temp.*/
                k=1;              /*variavel aux. utilizada p/ atualizar matriz da vel do
vent.*/

double  temp_historic1[vetor_size],          /*vetores de histórico de temperatura*/
                temp_historic2[vetor_size],
                temp_historic3[vetor_size],
                temp_historic4[vetor_size];


// ****** ELEVATOR MODULE **********8

//Verifica Sensor Terreo
```

```
    static int check_ground(void)
{
        daqErr = DIG_Prt_Config (1, 0, 0, 0);  /*Configura placa*/
        daqErr = DIG_In_Line (1, 0, 0, &instate0);
        return (instate0);

}

//Verifica Sensor 1.andar

static int check_first(void)
{
       daqErr = DIG_Prt_Config (1, 0, 0, 0);  /*Configura placa*/
       daqErr = DIG_In_Line (1, 0, 1, &instate1);
        return (instate1);
}


//Verifica Sensor 2.andar

static int check_second(void)
{
       daqErr = DIG_Prt_Config (1, 0, 0, 0);  /*Configura placa*/
       daqErr = DIG_In_Line (1, 0, 2, &instate2);
        return (instate2);
}

//Atualiza Sensores


static int update_sensor(void)
{
       sensor0 = check_ground();
       sensor1 = check_first();
       sensor2 = check_second();
       daqErr = DIG_Prt_Config (1, 0, 0, 0);  /*Configura placa*/
       daqErr = DIG_In_Line (1, 0, 5, &presenca_elev); /*verifica presenca no elev */

       return 0;
}

// Atualiza Leds dos andares do elevador
static int update_leds(void)
{
       SetCtrlVal (control, CONTROL_GROUND_FLOOR, sensor0);
       SetCtrlVal (control, CONTROL_FIRST_FLOOR, sensor1);
       SetCtrlVal (control, CONTROL_SECOND_FLOOR, sensor2);
       SetCtrlVal (control, CONTROL_OCCUPIED, presenca_elev);
       return 0;
```

```
    }

// Inicializa Elevador
static int initialize(void)
{

        int floor;
        daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
        daqErr = DIG_Out_Line (1, 0, 3, CW);    /*porta 3 - sent. hor*/

                update_sensor();
            /*verifica sensores elev.*/
        while (sensor0+sensor1+sensor2==0)  /*move elevador ate prox. piso (CW)*/
            {
              daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
              daqErr = DIG_Out_Line (1, 0, 4, enable); /*porta 4 - inicia mov.*/
              update_sensor();
            }

        daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
        daqErr = DIG_Out_Line (1, 0, 4, disable); /*porta 4 - interrompe mov.*/
         update_leds();

        if (sensor0==1)
            {
            floor = 0;
            }

       if (sensor1==1)
            {
            floor = 1;
            }

       if (sensor2==1)
            {
            floor = 2;
            }

        daqErr = DIG_Prt_Config (1, 0, 0, 1);
       daqErr = DIG_Out_Line (1, 0, 4, disable);
        return floor;

}

// Finaliza Controle
int CVICALLBACK Shutdown (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
       switch (event)
```

```
                {
            case EVENT_COMMIT:
                    QuitUserInterface (0);
                    break;
            case EVENT_RIGHT_CLICK:

                    break;
            }
        return 0;
}


// Move para primeiro andar

int CVICALLBACK go_first (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
        switch (event)
            {
            case EVENT_COMMIT:
                    GetCtrlVal (CONTROL, CONTROL_POWER, &powerstate);

                if (actual_floor>1 & powerstate==ON)
                        {
                                daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura
placa*/
                            daqErr = DIG_Out_Line (1, 0, 3, CW);    /*porta 3 - sent.
hor*/

                                while (sensor1==0)
                                    {
                                     daqErr = DIG_Prt_Config (1, 0, 0, 1);
/*Configura placa*/
                                     daqErr = DIG_Out_Line (1, 0, 4, enable);
/*porta 4 - inicia mov.*/
                                     update_sensor();
                                     update_leds();
                                    }
                                 daqErr = DIG_Prt_Config (1, 0, 0, 1);
/*Configura placa*/
                                 daqErr = DIG_Out_Line (1, 0, 4, disable); /*porta
4 - interrompe mov.*/
                                 update_sensor();
                                 update_leds();
                                 actual_floor =1;
                        }

                if (actual_floor<1 & powerstate==ON)
```

```
                                        {
                                          daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura
placa*/
                                          daqErr = DIG_Out_Line (1, 0, 3, CCW);    /*porta
3 - sent. ahor*/

                                          while (sensor1==0)
                                                {
                                                  daqErr = DIG_Prt_Config (1, 0, 0, 1);
/*Configura placa*/
                                                  daqErr = DIG_Out_Line (1, 0, 3, CCW);
/*porta 3 - sent. hor*/
                                                  daqErr = DIG_Out_Line (1, 0, 4, enable);
/*porta 4 - inicia mov.*/
                                                  update_sensor();
                                              update_leds();
                                                }

                                          update_sensor();
                                          update_leds();
                                          actual_floor =1;
                                        }
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        SetCtrlVal (CONTROL, CONTROL_NUMERIC, actual_floor);
        daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
        daqErr = DIG_Out_Line (1, 0, 4, disable); /*porta 4 - interrompe mov.*/
        return 0;
}

// move para andar térreo

int CVICALLBACK go_ground (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                        GetCtrlVal (CONTROL, CONTROL_POWER, &powerstate);

                        if (actual_floor>0 & powerstate==ON)
                                {
                                        daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura
placa*/
                                        daqErr = DIG_Out_Line (1, 0, 3, CW);    /*porta 3
- sent. hor*/
```

```
                                        while (sensor0==0)
                                                {
                                                 daqErr = DIG_Prt_Config (1, 0, 0, 1);
/*Configura placa*/
                                                 daqErr = DIG_Out_Line (1, 0, 4, enable);
/*porta 4 - inicia mov.*/
                                                 update_sensor();
                                            update_leds();
                                                }

                                        update_sensor();
                                        update_leds();
                                        actual_floor =0;
                                }

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        SetCtrlVal (CONTROL, CONTROL_NUMERIC, actual_floor);
        daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
        daqErr = DIG_Out_Line (1, 0, 4, disable); /*porta 4 - interrompe mov.*/
        return 0;
}

//Move para segundo andar

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

                        GetCtrlVal (CONTROL, CONTROL_POWER, &powerstate);
                        daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
                        daqErr = DIG_Out_Line (1, 0, 3, CCW);    /*porta 3 - sent.
a.hor*/
                        if (actual_floor<2 & powerstate==ON)
                                {
                                        while (sensor2==0)
                        {
                daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
                daqErr = DIG_Out_Line (1, 0, 3, CCW);    /*porta 3 - sent. a.hor*/
                 daqErr = DIG_Out_Line (1, 0, 4, enable); /*porta 4 - inicia mov.*/
                        update_sensor();
                        update_leds();
```

```
                    }

                update_sensor();
                 update_leds();
                 actual_floor =2;
                    }
                break;
            case EVENT_RIGHT_CLICK:

                break;
            }
        SetCtrlVal (CONTROL, CONTROL_NUMERIC, actual_floor);
         daqErr = DIG_Prt_Config (1, 0, 0, 1);   /*Configura placa*/
        daqErr = DIG_Out_Line (1, 0, 4, disable); /*porta 4 - interrompe mov.*/
        return 0;
}


// Atualiza display de status do elevador
static int elevator_status_display(int status)
{ if (status == 0)
        {
         ResetTextBox (control, CONTROL_STATUS, "NOT READY");
        }

 if (status == 1)
  {
        ResetTextBox (control, CONTROL_STATUS, "NOT READY");
  }

 if (status == 2)
  {
   ResetTextBox (control, CONTROL_STATUS, "EMERGENCY");
  }
 return 0;
}



//Permite ou não movimentação do elevador

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

            GetCtrlVal (CONTROL, CONTROL_POWER, &powerstate);
```

```
                    if (powerstate==OFF)
                {
        daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
        daqErr = DIG_Out_Line (1, 0, 4, disable); /*porta 4 - interrompe mov.*/
                                update_sensor();
                                update_leds();
                                elevator_status_display(0);

                }

                        if (powerstate==ON)
                                {
                                actual_floor = initialize();
                                elevator_status_display(1);
                                SetCtrlVal (CONTROL,
CONTROL_NUMERIC,actual_floor);
                                }

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}


//chama e fecha painel de teste

int CVICALLBACK display_testing (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                        DisplayPanel (testing);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

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

```
                    case EVENT_COMMIT:
                        HidePanel (testing);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}


//**** END ELEVATOR ******************


//**** MAIN MODULE**********************

int main (int argc, char *argv[])
{
        short placa;
        if (InitCVIRTE (0, argv, 0) == 0)     /* Needed if linking in external compiler;
harmless otherwise */
                return -1;      /* out of memory */
        if ((testing = LoadPanel (0, "Supervisor.uir", TESTING)) < 0)
                return -1;
        if ((control = LoadPanel (0, "supervisor.uir", CONTROL)) < 0)
                return -1;
        if ((tempRoom1 = LoadPanel (0, "supervisor.uir", TEMP_ROOM1)) < 0)
                return -1;
        if ((tempRoom2 = LoadPanel (0, "supervisor.uir", TEMP_ROOM2)) < 0)
                return -1;
        if ((tempRoom3 = LoadPanel (0, "supervisor.uir", TEMP_ROOM3)) < 0)
                return -1;
        if ((tempRoom4 = LoadPanel (0, "supervisor.uir", TEMP_ROOM4)) < 0)
                return -1;


        DisplayPanel (control);
        DisplayPanel (testing);
        RunUserInterface ();
        daqErr = Init_DA_Brds (1, &placa);          /*Inicializa placa*/
        daqErr = DIG_Prt_Config (1, 0, 0, 1);  /*Configura placa*/
        daqErr = DIG_Out_Line (1, 0, 4, disable); /*porta 4 - mov. desabilitado*/

        /*inicializa velocidade dos ventiladores*/
        GetCtrlVal (tempRoom1, TEMP_ROOM1_FAN_SPEED1, &fanspeed[1]);
        GetCtrlVal (tempRoom2, TEMP_ROOM2_FAN_SPEED2, &fanspeed[2]);
        GetCtrlVal (tempRoom3, TEMP_ROOM3_FAN_SPEED3, &fanspeed[3]);
        GetCtrlVal (tempRoom4, TEMP_ROOM4_FAN_SPEED4, &fanspeed[4]);
```

```
        return 0;
}

// ********* END MAIN *********

// *** PRESENCE MODULE **************

//Verifica presenca nas salas

static int check_presence(void)
{
                GetCtrlVal (testing, TESTING_PRESENCE1, &occupation[1]);
                GetCtrlVal (testing, TESTING_PRESENCE2, &occupation[2]);
                GetCtrlVal (testing, TESTING_PRESENCE3, &occupation[3]);
                GetCtrlVal (testing, TESTING_PRESENCE4, &occupation[4]);
                GetCtrlVal (testing, TESTING_PRESENCE5, &occupation[5]);

        return(0);

}

// Atualiza Leds de presença

static int update_leds_presence(void)
{
        SetCtrlVal (control, CONTROL_PRESENCE_LED1, occupation[1]);
        SetCtrlVal (control, CONTROL_PRESENCE_LED2, occupation[2]);
        SetCtrlVal (control, CONTROL_PRESENCE_LED3, occupation[3]);
        SetCtrlVal (control, CONTROL_PRESENCE_LED4, occupation[4]);
        SetCtrlVal (control, CONTROL_PRESENCE_LED5, occupation[5]);

 return 0;
}

//Verifica permissao de presenca nas salas

static int check_permission(void)
{
                GetCtrlVal (control, CONTROL_ENABLE1, &permission[1]);
                GetCtrlVal (control, CONTROL_ENABLE2, &permission[2]);
                GetCtrlVal (control, CONTROL_ENABLE3, &permission[3]);
                GetCtrlVal (control, CONTROL_ENABLE4, &permission[4]);
                GetCtrlVal (control, CONTROL_ENABLE5, &permission[5]);

        return(0);

}
```

```
// Verifica se existe presença em alguma sala não habilitada, neste caso dispara alarme.

static int display_emergency(void)
{
 int i, intruder = 0;
 for (i=1; i<6; i++)
        {
         if(permission[i]==0 & occupation[i]==1)
               {intruder = 1;}
        }
        if (intruder == 1)
               {
               SetCtrlVal (control, CONTROL_EMERGENCY, intruder);
               ResetTextBox (control, CONTROL_EMERGENCY_TEXT, "
INTRUDER");


               }
       if (intruder == 0)
               {
               SetCtrlVal (control, CONTROL_EMERGENCY, intruder);
               ResetTextBox (control, CONTROL_EMERGENCY_TEXT, "       ");
               }

  return(0);
 }

int CVICALLBACK timer_update (int panel, int control, int event,
               void *callbackData, int eventData1, int eventData2)
{
       switch (event)
               {
               case EVENT_TIMER_TICK:
                       check_presence();
                       update_leds_presence();
                       check_permission();
                       display_emergency();

                       break;
               }
       return 0;
}

//**** END PRESENCE ********

//***** TEMPERATURE MODULE**********
```

```
int CVICALLBACK show_room1 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
        DisplayPanel (tempRoom1);

            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}

int CVICALLBACK close_room1 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            HidePanel (tempRoom1);

            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}


int CVICALLBACK show_room2 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
          DisplayPanel (tempRoom2);
            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}

int CVICALLBACK close_room2 (int panel, int control, int event,
```

```
                    void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                        HidePanel (tempRoom2);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}


int CVICALLBACK show_room3 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                        DisplayPanel (tempRoom3);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK close_room3 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                        HidePanel (tempRoom3);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK show_room4 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
```

```
                {
                case EVENT_COMMIT:
                        DisplayPanel (tempRoom4);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK close_room4 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                        HidePanel (tempRoom4);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}


int CVICALLBACK send_temp1 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK send_fan_speed1 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
```

```
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}


// Verifica qual sala esta ativa para atualizacao
static int selected_room(void)
{
        int room_to_update;
        GetCtrlVal (testing, TESTING_ACTUAL_ROOM, &room_to_update);

        return(room_to_update);

}



// Verifica temperatura das salas
static int check_temp(int position)
{   int room;
        int j;          /*variavel auxiliar para percorrer o vetor*/

        if (position<vetor_size)
                {  room = selected_room();     /* verifica qual sala deve ser atualizada */

                        //********sala 1*******
                        if (room == 1)
                         {
GetCtrlVal (testing, TESTING_ROOM_TEMP, &temp_historic1[position]);
 SetCtrlVal (control, CONTROL_ACTUAL_TEMP1, temp_historic1[position]);
 SetCtrlVal (tempRoom1, TEMP_ROOM1_ACTUAL_TEMP1,
temp_historic1[position]);

                         }
       else /*se outra sala for a selecionada, mantem temp.atual */
                        {
                         temp_historic1[position]=temp_historic1[position-1];
                        }

                        //********sala 2*******
                        if (room == 2)
                {
GetCtrlVal (testing, TESTING_ROOM_TEMP, &temp_historic2[position]);
SetCtrlVal (control, CONTROL_ACTUAL_TEMP2, temp_historic2[position] );
```

```
        SetCtrlVal (tempRoom2, TEMP_ROOM2_ACTUAL_TEMP2,
    temp_historic2[position]);
                                }
                               else
                                {
                                 temp_historic2[position]=temp_historic2[position-1];
                                }

                                //********sala 3*******
                               if (room == 3)
                                {
           GetCtrlVal (testing, TESTING_ROOM_TEMP, &temp_historic3[position]);
           SetCtrlVal (control, CONTROL_ACTUAL_TEMP3, temp_historic3[position]);
                                    SetCtrlVal (tempRoom3,
    TEMP_ROOM3_ACTUAL_TEMP3, temp_historic3[position]);
                                }
                               else
                                {
                                 temp_historic3[position]=temp_historic3[position-1];
                                }

                                //********sala 4*******
                               if (room == 4)
                                {
     GetCtrlVal (testing, TESTING_ROOM_TEMP, &temp_historic4[position]);
    SetCtrlVal (control, CONTROL_ACTUAL_TEMP4, temp_historic4[position]);
     SetCtrlVal (tempRoom4, TEMP_ROOM4_ACTUAL_TEMP4,
    temp_historic4[position]);
                                }
                               else
                                {
                                 temp_historic4[position]=temp_historic4[position-1];
                                }

                }
          else                  /* se vetor já estiver cheio, temp. entra no final*/
                {
                                room = selected_room();         /* verifica qual sala deve ser
    atualizada */

                                        /*** sala 1 ******/
                                        if (room == 1)
                                        {
                                                for (j=0; j<vetor_size-1; j++)
                                                  {
                                                  temp_historic1[j]=temp_historic1[j+1];
                                                  }
    GetCtrlVal (testing, TESTING_ROOM_TEMP, &temp_historic1[vetor_size-1]);
    SetCtrlVal (control, CONTROL_ACTUAL_TEMP1, temp_historic1[vetor_size-1]);
```

```
        SetCtrlVal (tempRoom1, TEMP_ROOM1_ACTUAL_TEMP1,
temp_historic1[vetor_size-1]);
                                }

                                /*** sala 2 ******/
                                if (room == 2)
                                {
                                        for (j=0; j<vetor_size-1; j++)
                                          {
                                          temp_historic2[j]=temp_historic2[j+1];
                                          }
GetCtrlVal (testing, TESTING_ROOM_TEMP, &temp_historic2[vetor_size-1]);
SetCtrlVal (control, CONTROL_ACTUAL_TEMP2, temp_historic2[vetor_size-1]);
SetCtrlVal (tempRoom2, TEMP_ROOM2_ACTUAL_TEMP2,
temp_historic2[vetor_size-1]);

                                }

                                        /*** sala 3 ******/
                                if (room == 3)
                                {
                                        for (j=0; j<vetor_size-1; j++)
                                          {
                                          temp_historic3[j]=temp_historic3[j+1];
                                          }
GetCtrlVal (testing, TESTING_ROOM_TEMP, &temp_historic3[vetor_size-1]);
SetCtrlVal (control, CONTROL_ACTUAL_TEMP3, temp_historic3[vetor_size-1]);
SetCtrlVal (tempRoom3, TEMP_ROOM3_ACTUAL_TEMP3,
temp_historic3[vetor_size-1]);

                                }

                                        /*** sala 4 ******/
                                if (room == 4)
                                {
                                        for (j=0; j<vetor_size-1; j++)
                                          {
                                          temp_historic4[j]=temp_historic4[j+1];
                                          }
GetCtrlVal (testing, TESTING_ROOM_TEMP, &temp_historic4[vetor_size-1]);
SetCtrlVal (control, CONTROL_ACTUAL_TEMP4, temp_historic4[vetor_size-1]);
SetCtrlVal (tempRoom4, TEMP_ROOM4_ACTUAL_TEMP4,
temp_historic4[vetor_size-1]);

                                }

        }

    return 0;
```

```
        }

    //Verifica status de fumaça e incendio das salas
    static int check_smoke_fire(void)
    {  int k,m;
          int room;

     room = selected_room(); /*Verifica qual sala esta ativa*/

     /*Executa leitura dos dados para cada sala*/
     for (k=1; k<5; k++)
          {
           if (room==k)
                        {
     GetCtrlVal (testing, TESTING_ROOM_SMOKE, &smoke[k]);
     GetCtrlVal (testing, TESTING_ROOM_FIRE, &fire[k]);
                        }
          }
     /*Atualiza Leds dos paineis de detalhes*/
     SetCtrlVal (tempRoom1, TEMP_ROOM1_SMOKE1, smoke[1]);
     SetCtrlVal (tempRoom2, TEMP_ROOM2_SMOKE2, smoke[2]);
     SetCtrlVal (tempRoom3, TEMP_ROOM3_SMOKE3, smoke[3]);
     SetCtrlVal (tempRoom4, TEMP_ROOM4_SMOKE4, smoke[4]);

     SetCtrlVal (tempRoom1, TEMP_ROOM1_FIRE1, fire[1]);
     SetCtrlVal (tempRoom2, TEMP_ROOM2_FIRE2, fire[2]);
     SetCtrlVal (tempRoom3, TEMP_ROOM3_FIRE3, fire[3]);
     SetCtrlVal (tempRoom4, TEMP_ROOM4_FIRE4, fire[4]);


  /*Atualiza Leds do painel de controle*/
  //*** Sala 1****

    if(smoke[1] || fire[1] == 1)
    {
     SetCtrlVal (control, CONTROL_STATUS_TEMP1, ON);
    }
    else
    {
     SetCtrlVal (control, CONTROL_STATUS_TEMP1, OFF);
    }

 //*** Sala 2****

    if(smoke[2] || fire[2] == 1)
    {
     SetCtrlVal (control, CONTROL_STATUS_TEMP2, ON);
    }
    else
```

```
        {
     SetCtrlVal (control, CONTROL_STATUS_TEMP2, OFF);
    }

  //*** Sala 3****

    if(smoke[3] || fire[3] == 1)
    {
     SetCtrlVal (control, CONTROL_STATUS_TEMP3, ON);
    }
    else
    {
     SetCtrlVal (control, CONTROL_STATUS_TEMP3, OFF);
    }

 //*** Sala 4****

    if(smoke[4] || fire[4] == 1)
    {
     SetCtrlVal (control, CONTROL_STATUS_TEMP4, ON);
    }
    else
    {
     SetCtrlVal (control, CONTROL_STATUS_TEMP4, OFF);
    }


// Aciona controle de incendio para a sala que estiver com sinal de incendio
        if (fire[1]==1)
                {
                 SetCtrlVal (tempRoom1, TEMP_ROOM1_FIRE_PROCEDURE, ON);
                 fire_actions();
                }
        if (fire[2]==1)
                {
        SetCtrlVal (tempRoom2, TEMP_ROOM2_FIRE_PROCEDURE, ON);
                 fire_actions();
                }
        if (fire[3]==1)
                {
        SetCtrlVal (tempRoom3, TEMP_ROOM3_FIRE_PROCEDURE, ON);
                 fire_actions();
                }
       if (fire[4]==1)
                {
        SetCtrlVal (tempRoom4, TEMP_ROOM4_FIRE_PROCEDURE, ON);
                 fire_actions();
                }
```

```
    return 0;

    }


    // Loop de rotinas referentes à temperatura
    int CVICALLBACK timer2_update (int panel. int control, int event,
                    void *callbackData, int eventData1, int eventData2)
    {

            switch (event)
                    {
                    case EVENT_TIMER_TICK:

                            check_temp(i);
                            check_smoke_fire();

                                    i++;


                            break;
                    }
            return 0;
    }

    /*Realiza ações referentes ao procedimento de incendio*/
    static int fire_actions(void)
    {
     elevator_status_display(2);
     //initialize();
     SetCtrlVal (control, CONTROL_POWER. OFF);
    return 0;
    }



    // Procedimento executado em caso de incendio
    int CVICALLBACK fire_procedure (int panel. int control, int event,
                    void *callbackData, int eventData1, int eventData2)
    {  int fire_status[5];
      int p;

            switch (event)
                    {
                    case EVENT_COMMIT:
                    GetCtrlVal(tempRoom1,TEMP_ROOM1_FIRE_PROCEDURE,&fire_st
           atus[1]);
```

```
                    GetCtrlVal (tempRoom2, TEMP_ROOM2_FIRE_PROCEDURE,
&fire_status[2]);
                GetCtrlVal (tempRoom3, TEMP_ROOM3_FIRE_PROCEDURE,
&fire_status[3]);
                GetCtrlVal (tempRoom4, TEMP_ROOM4_FIRE_PROCEDURE,
&fire_status[4]);

                        for (p=1; p<5; p++)
                                {
                                fire[p]=fire_status[p];
                                }

                        if (fire_status[1]+fire_status[2]+fire_status[3]+fire_status[4]>=1)
                        {
                         fire_actions();
                        }
                        if (fire_status[1]+fire_status[2]+fire_status[3]+fire_status[4]==0)
                        {
                         elevator_status_display(0);
                        }

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}


// Plota histórico de temperatura e de velocidade do ventilador
int CVICALLBACK show_history1 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:

DeleteGraphPlot (tempRoom1, TEMP_ROOM1_GRAPH_TEMP1, -1,
                                                VAL_IMMEDIATE_DRAW);

PlotY (tempRoom1, TEMP_ROOM1_GRAPH_TEMP1, temp_historic1,
vetor_size,VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
VAL_SOLID, 1,VAL_RED);


DeleteGraphPlot (tempRoom1, TEMP_ROOM1_GRAPH_FAN1, -1,
VAL_IMMEDIATE_DRAW);
```

```
        PlotY (tempRoom1, TEMP_ROOM1_GRAPH_FAN1, fan_historic[1],
    vetor_fan_size, VAL_INTEGER, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
    VAL_SOLID, 1, VAL_RED);


                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK show_history2 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
DeleteGraphPlot (tempRoom2, TEMP_ROOM2_GRAPH_TEMP2, -1,
                                                        VAL_IMMEDIATE_DRAW);

PlotY (tempRoom2, TEMP_ROOM2_GRAPH_TEMP2, temp_historic2,
vetor_size,VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
VAL_SOLID, 1,   VAL_RED);

DeleteGraphPlot (tempRoom2, TEMP_ROOM2_GRAPH_FAN2, -1,
                                                        VAL_IMMEDIATE_DRAW);

PlotY (tempRoom2, TEMP_ROOM2_GRAPH_FAN2, fan_historic[2],
vetor_fan_size, VAL_INTEGER, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
VAL_SOLID, 1, VAL_RED);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK show_history3 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
DeleteGraphPlot (tempRoom3, TEMP_ROOM3_GRAPH_TEMP3, -
1,VAL_IMMEDIATE_DRAW);
```

```
        PlotY (tempRoom3, TEMP_ROOM3_GRAPH_TEMP3, temp_historic3,
    vetor_size, VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
    VAL_SOLID, 1, VAL_RED);

    DeleteGraphPlot (tempRoom3, TEMP_ROOM3_GRAPH_FAN3, -1,
                                                    VAL_IMMEDIATE_DRAW);

    PlotY (tempRoom3, TEMP_ROOM3_GRAPH_FAN3, fan_historic[3],
     vetor_fan_size, VAL_INTEGER, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
    VAL_SOLID, 1, VAL_RED);
                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK show_history4 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
DeleteGraphPlot (tempRoom4, TEMP_ROOM4_GRAPH_TEMP4, -1,
                                                    VAL_IMMEDIATE_DRAW);

PlotY (tempRoom4, TEMP_ROOM4_GRAPH_TEMP4, temp_historic4, vetor_size,
VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_RED);

DeleteGraphPlot (tempRoom4, TEMP_ROOM4_GRAPH_FAN4, -1,
                                                    VAL_IMMEDIATE_DRAW);

PlotY (tempRoom4, TEMP_ROOM4_GRAPH_FAN4, fan_historic[4],
vetor_fan_size, VAL_INTEGER, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
                                VAL_SOLID, 1, VAL_RED);

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}


int CVICALLBACK send_temp4 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
```

```
        {
          switch (event)
                {
                case EVENT_COMMIT:

                      break;
                case EVENT_RIGHT_CLICK:

                      break;
                }
          return 0;
 }

 int CVICALLBACK send_fan_speed4 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
 {
        switch (event)
                {
                case EVENT_COMMIT:

                      break;
                case EVENT_RIGHT_CLICK:

                      break;
                }
        return 0;
 }


 int CVICALLBACK send_temp3 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:

                      break;
                case EVENT_RIGHT_CLICK:

                      break;
                }
        return 0;
}

int CVICALLBACK send_fan_speed3 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
```

```
                {
                case EVENT_COMMIT:

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK send_temp2 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}

int CVICALLBACK send_fan_speed2 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:

                        break;
                case EVENT_RIGHT_CLICK:

                        break;
                }
        return 0;
}


///**** Modulo do Ventilador*******////

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

```
                    case EVENT_TIMER_TICK:
                        check_fan(k);

                        k++;
                        break;
                }
        return 0;
}

// Verifica status dos ventiladores e atualiza velocidade e dados históricos
static int check_fan(int position_fan)
{  int presenca1,habilita1,presenca2,habilita2,presenca3,habilita3,presenca4,habilita4;
   int fan_speed;
   int room_fan;
   double actual_temp1,actual_temp2,actual_temp3,actual_temp4;
   double temp_setpoint1,temp_setpoint2,temp_setpoint3,temp_setpoint4;


/***sala 1****/

GetCtrlVal (testing, TESTING_PRESENCE1, &presenca1); //Recebe dados da sala
GetCtrlVal (control, CONTROL_ENABLE1, &habilita1);   // presenca e habilitação

        /* Se sala desabilitada ou vazia, ventilador vai para velocidade padrao*/
        if (presenca1==0 || habilita1 ==0)
                {
GetCtrlVal (tempRoom1, TEMP_ROOM1_FAN_SPEED1, &fan_speed);
                 update_fan_matrix(1,position_fan,fan_speed);
                }

        /* Se sala habilitada e com presenca, velocidade ajustada em funcao da temp. */
        if (presenca1==1 && habilita1 ==1)
                {
GetCtrlVal (tempRoom1, TEMP_ROOM1_TEMP_ROOM1, &temp_setpoint1);
GetCtrlVal (tempRoom1, TEMP_ROOM1_ACTUAL_TEMP1, &actual_temp1);

                /* Se temp = setpoint mantém velocidade */
                if (abs(temp_setpoint1-actual_temp1)<=0.5)
                {
                 update_fan_matrix(1,position_fan,fanspeed[1]);
                }

                /* Se temp > setpoint aumenta vel. ate limite de 10 */
                if ((actual_temp1-temp_setpoint1)>0.5)
                {
                 if(fanspeed[1]==10)
                        {
                        update_fan_matrix(1,position_fan,fanspeed[1]);
                        }
```

```
                else
                 {
                 update_fan_matrix(1,position_fan,fanspeed[1]+1);
                 }
                }

                /* Se temp < setpoint diminui vel. ate limite de 0 */
                if ((actual_temp1-temp_setpoint1)<-0.5)
                {
                 if(fanspeed[1]==0)
                      {
                       update_fan_matrix(1,position_fan,fanspeed[1]);
                      }
                 else
                 {
                 update_fan_matrix(1,position_fan,fanspeed[1]-1);
                 }
                }

               }

//***sala 2****/

GetCtrlVal (testing, TESTING_PRESENCE2, &presenca2);  //Recebe dados da sala
GetCtrlVal (control, CONTROL_ENABLE2, &habilita2);   // presenca e habilitação

      /* Se sala desabilitada ou vazia, ventilador vai para velocidade padrao*/
      if (presenca2==0 || habilita2 ==0)
            {
      GetCtrlVal (tempRoom2, TEMP_ROOM2_FAN_SPEED2, &fan_speed);
             update_fan_matrix(2,position_fan,fan_speed);
            }

      /* Se sala habilitada e com presenca, velocidade ajustada em funcao da temp. */
      if (presenca2==1 && habilita2 ==1)
            {
GetCtrlVal (tempRoom2, TEMP_ROOM2_TEMP_ROOM2, &temp_setpoint2);
GetCtrlVal (tempRoom2, TEMP_ROOM2_ACTUAL_TEMP2, &actual_temp2);

                /* Se temp = setpoint mantém velocidade */
                if (abs(temp_setpoint2-actual_temp2)<=0.5)
                {
                 update_fan_matrix(2,position_fan,fanspeed[2]);
                }

                /* Se temp > setpoint aumenta vel. ate limite de 10 */
                if ((actual_temp2-temp_setpoint2)>0.5)
                {
                 if(fanspeed[2]==10)
```

```
                    {
                    update_fan_matrix(2,position_fan,fanspeed[2]);
                    }
                  else
                  {
                  update_fan_matrix(2,position_fan,fanspeed[2]+1);
                  }
                 }

                 /* Se temp < setpoint diminui vel. ate limite de 0 */
                 if ((actual_temp2-temp_setpoint2)<-0.5)
                 {
                  if(fanspeed[2]==0)
                        {
                        update_fan_matrix(2,position_fan,fanspeed[2]);
                        }
                  else
                  {
                  update_fan_matrix(2,position_fan,fanspeed[2]-1);
                  }
                 }

                }

/***sala 3****/
GetCtrlVal (testing, TESTING_PRESENCE3, &presenca3);  //Recebe dados da sala
GetCtrlVal (control, CONTROL_ENABLE3, &habilita3);   // presenca e habilitação

        /* Se sala desabilitada ou vazia, ventilador vai para velocidade padrao*/
        if (presenca3==0 || habilita3 ==0)
                {
                 GetCtrlVal (tempRoom3, TEMP_ROOM3_FAN_SPEED3,
&fan_speed);
                 update_fan_matrix(3,position_fan,fan_speed);
                }

       /* Se sala habilitada e com presenca, velocidade ajustada em funcao da temp. */
       if (presenca3==1 && habilita3 ==1)
                {
GetCtrlVal (tempRoom3, TEMP_ROOM3_TEMP_ROOM3, &temp_setpoint3);
GetCtrlVal (tempRoom3, TEMP_ROOM3_ACTUAL_TEMP3, &actual_temp3);

                /* Se temp = setpoint mantém velocidade */
                if (abs(temp_setpoint3-actual_temp3)<=0.5)
                {
                 update_fan_matrix(3,position_fan,fanspeed[3]);
                }

                /* Se temp > setpoint aumenta vel. ate limite de 10 */
```

```
                if ((actual_temp3-temp_setpoint3)>0.5)
            {
             if(fanspeed[3]==10)
                  {
                   update_fan_matrix(3,position_fan,fanspeed[3]);
                  }
             else
             {
             update_fan_matrix(3,position_fan,fanspeed[3]+1);
             }
            }

            /* Se temp < setpoint diminui vel. ate limite de 0 */
            if ((actual_temp3-temp_setpoint3)<-0.5)
            {
             if(fanspeed[3]==0)
                  {
                   update_fan_matrix(3,position_fan,fanspeed[3]);
                  }
             else
             {
             update_fan_matrix(3,position_fan,fanspeed[3]-1);
             }
            }

          }

/***sala 4****/

GetCtrlVal (testing, TESTING_PRESENCE4. &presenca4);  //Recebe dados da sala
GetCtrlVal (control, CONTROL_ENABLE4. &habilita4);   // presenca e habilitação

      /* Se sala desabilitada ou vazia, ventilador vai para velocidade padrao*/
      if (presenca4==0 || habilita4 ==0)
            {
      GetCtrlVal (tempRoom4, TEMP_ROOM4_FAN_SPEED4, &fan_speed);
             update_fan_matrix(4,position_fan,fan_speed);
            }

      /* Se sala habilitada e com presenca, velocidade ajustada em funcao da temp. */
      if (presenca4==1 && habilita4 ==1)
            {
GetCtrlVal (tempRoom4, TEMP_ROOM4_TEMP_ROOM4, &temp_setpoint4);
GetCtrlVal (tempRoom4, TEMP_ROOM4_ACTUAL_TEMP4, &actual_temp4);

            /* Se temp = setpoint mantém velocidade */
            if (abs(temp_setpoint4-actual_temp4)<=0.5)
            {
             update_fan_matrix(4,position_fan,fanspeed[4]);
```

```
                }

              /* Se temp > setpoint aumenta vel. ate limite de 10 */
              if ((actual_temp4-temp_setpoint4)>0.5)
              {
               if(fanspeed[4]==10)
                    {
                     update_fan_matrix(4,position_fan,fanspeed[4]);
                    }
               else
               {
               update_fan_matrix(4,position_fan,fanspeed[4]+1);
               }
              }

              /* Se temp < setpoint diminui vel. ate limite de 0 */
              if ((actual_temp4-temp_setpoint4)<-0.5)
              {
               if(fanspeed[4]==0)
                    {
                     update_fan_matrix(4,position_fan,fanspeed[4]);
                    }
               else
               {
               update_fan_matrix(4,position_fan,fanspeed[4]-1);
               }
              }

             }

 return(0);

}


/*funcao que atualiza matriz de velocidade do ventilador*/

static int update_fan_matrix(int room,int column, int speed)
{  int p;  /*variavel auxiliar para percorrer vetor*/

        if (column<vetor_fan_size)
        {
        fan_historic[room][column] = speed;
        fanspeed[room]=speed;
        }
        else
        {
          for (p=0; p<vetor_fan_size-1; p++)
```

```
                {
            fan_historic[room][p]=fan_historic[room][p+1];
            }
                fan_historic[room][vetor_fan_size-1]=speed;
                fanspeed[room]=speed;
        }

 return 0;
}
```

Apêndice II.2 Listagem do programa Supervisor.H

Este arquivo contém as definições das funções utilizadas no programa. As funções chamadas através da iteração do usuário e de todos os elementos presentes na interface com o usuário.

```
/*****************************************************************************/
/* LabWindows/CVI User Interface Resource (UIR) Include File                 */
/* Copyright (c) National Instruments 2001. 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  CONTROL                         1
#define  CONTROL_QUIT                    2       /* callback function: Shutdown */
#define  CONTROL_SECOND_FLOOR            3
#define  CONTROL_FIRST_FLOOR             4
#define  CONTROL_GROUND_FLOOR            5
#define  CONTROL_OCCUPIED                6
#define  CONTROL_POWER                   7       /* callback function: power_function */
#define  CONTROL_CALL_SECOND             8       /* callback function: go_second */
#define  CONTROL_CALL_FIRST              9       /* callback function: go_first */
#define  CONTROL_CALL_GROUND             10      /* callback function: go_ground
*/
#define  CONTROL_NUMERIC                 11
#define  CONTROL_PRESENCE_LED1           12
#define  CONTROL_PRESENCE_LED2           13
#define  CONTROL_PRESENCE_LED3           14
#define  CONTROL_PRESENCE_LED4           15
#define  CONTROL_PRESENCE_LED5           16
#define  CONTROL_ENABLE5                 17
#define  CONTROL_ENABLE4                 18
#define  CONTROL_ENABLE3                 19
#define  CONTROL_ENABLE2                 20
```

```
#define CONTROL_ENABLE1 21
#define CONTROL_EMERGENCY 22
#define CONTROL_EMERGENCY_TEXT 23
#define CONTROL_DETAIL1 24 /* callback function: show_room1 */
#define CONTROL_TESTING_BUTTON 25 /* callback function:
display_testing */
#define CONTROL_STATUS_TEMP1 26
#define CONTROL_DETAIL2 27 /* callback function: show_room2 */
#define CONTROL_STATUS_TEMP2 28
#define CONTROL_DETAIL4 29 /* callback function: show_room4 */
#define CONTROL_DETAIL3 30 /* callback function: show_room3 */
#define CONTROL_STATUS_TEMP3 31
#define CONTROL_STATUS_TEMP4 32
#define CONTROL_ACTUAL_TEMP2 33
#define CONTROL_ACTUAL_TEMP1 34
#define CONTROL_ACTUAL_TEMP3 35
#define CONTROL_ACTUAL_TEMP4 36
#define CONTROL_STATUS 37
#define CONTROL_TEXTMSG 38
#define CONTROL_DECORATION_2 39
#define CONTROL_TEXTMSG_2 40
#define CONTROL_TEXTMSG_3 41
#define CONTROL_TEXTMSG_4 42
#define CONTROL_TEXTMSG_5 43
#define CONTROL_TEXTMSG_6 44
#define CONTROL_DECORATION_3 45
#define CONTROL_DECORATION 46
#define CONTROL_DECORATION_4 47
#define CONTROL_DECORATION_5 48
#define CONTROL_DECORATION_6 49
#define CONTROL_DECORATION_7 50
#define CONTROL_DECORATION_8 51
#define CONTROL_DECORATION_10 52
#define CONTROL_DECORATION_11 53
#define CONTROL_DECORATION_12 54
#define CONTROL_DECORATION_13 55
#define CONTROL_TEXTMSG_7 56
#define CONTROL_DECORATION_9 57
#define CONTROL_TIMER 58 /* callback function: timer_update */
#define CONTROL_TIMER_2 59 /* callback function: timer2_update */
#define CONTROL_TIMER_3 60 /* callback function: fan_timer */

#define TEMP_ROOM1 2
#define TEMP_ROOM1_TEMP_ROOM1 2 /* callback function:
send_temp1 */
#define TEMP_ROOM1_ACTUAL_TEMP1 3
#define TEMP_ROOM1_CLOSE1 4 /* callback function: close_room1 */
#define TEMP_ROOM1_FIRE1 5
#define TEMP_ROOM1_SMOKE1 6
```

```
    #define TEMP_ROOM1_FIRE_PROCEDURE      7      /* callback function:
fire_procedure */
#define TEMP_ROOM1_FAN_SPEED1          8      /* callback function:
send_fan_speed1 */
#define TEMP_ROOM1_GRAPH_FAN1          9
#define TEMP_ROOM1_GRAPH_TEMP1         10
#define TEMP_ROOM1_COMMANDBUTTON       11     /* callback function:
show_history1 */
#define TEMP_ROOM1_DECORATION          12
#define TEMP_ROOM1_DECORATION_2        13
#define TEMP_ROOM1_DECORATION_3        14
#define TEMP_ROOM1_DECORATION_4        15

#define TEMP_ROOM2                     3
#define TEMP_ROOM2_TEMP_ROOM2          2      /* callback function:
send_temp2 */
#define TEMP_ROOM2_ACTUAL_TEMP2        3
#define TEMP_ROOM2_CLOSE2              4      /* callback function: close_room2 */
#define TEMP_ROOM2_FIRE2               5
#define TEMP_ROOM2_SMOKE2              6
#define TEMP_ROOM2_FIRE_PROCEDURE      7      /* callback function:
fire_procedure */
#define TEMP_ROOM2_FAN_SPEED2          8      /* callback function:
send_fan_speed2 */
#define TEMP_ROOM2_GRAPH_FAN2          9
#define TEMP_ROOM2_GRAPH_TEMP2         10
#define TEMP_ROOM2_COMMANDBUTTON       11     /* callback function:
show_history2 */
#define TEMP_ROOM2_DECORATION          12
#define TEMP_ROOM2_DECORATION_2        13
#define TEMP_ROOM2_DECORATION_3        14
#define TEMP_ROOM2_DECORATION_4        15

#define TEMP_ROOM3                     4
#define TEMP_ROOM3_TEMP_ROOM3          2      /* callback function:
send_temp3 */
#define TEMP_ROOM3_ACTUAL_TEMP3        3
#define TEMP_ROOM3_CLOSE3              4      /* callback function: close_room3 */
#define TEMP_ROOM3_FIRE3               5
#define TEMP_ROOM3_SMOKE3              6
#define TEMP_ROOM3_FIRE_PROCEDURE      7      /* callback function:
fire_procedure */
#define TEMP_ROOM3_FAN_SPEED3          8      /* callback function:
send_fan_speed3 */
#define TEMP_ROOM3_GRAPH_FAN3          9
#define TEMP_ROOM3_GRAPH_TEMP3         10
#define TEMP_ROOM3_COMMANDBUTTON       11     /* callback function:
show_history3 */
#define TEMP_ROOM3_DECORATION          12
```

```
#define TEMP_ROOM3_DECORATION_2        13
#define TEMP_ROOM3_DECORATION_3        14
#define TEMP_ROOM3_DECORATION_4        15

#define TEMP_ROOM4                     5
#define TEMP_ROOM4_TEMP_ROOM4          2      /* callback function:
send_temp4 */
#define TEMP_ROOM4_ACTUAL_TEMP4        3
#define TEMP_ROOM4_CLOSE4              4      /* callback function: close_room4 */
#define TEMP_ROOM4_FIRE4               5
#define TEMP_ROOM4_SMOKE4              6
#define TEMP_ROOM4_FIRE_PROCEDURE      7      /* callback function:
fire_procedure */
#define TEMP_ROOM4_FAN_SPEED4          8      /* callback function:
send_fan_speed4 */
#define TEMP_ROOM4_GRAPH_FAN4          9
#define TEMP_ROOM4_GRAPH_TEMP4         10
#define TEMP_ROOM4_COMMANDBUTTON       11     /* callback function:
show_history4 */
#define TEMP_ROOM4_DECORATION          12
#define TEMP_ROOM4_DECORATION_2        13
#define TEMP_ROOM4_DECORATION_3        14
#define TEMP_ROOM4_DECORATION_4        15

#define TESTING                        6
#define TESTING_PRESENCE5              2
#define TESTING_PRESENCE4              3
#define TESTING_CLOSE_TEST             4      /* callback function: close_testing */
#define TESTING_PRESENCE3              5
#define TESTING_PRESENCE2              6
#define TESTING_PRESENCE1              7
#define TESTING_ROOM_TEMP              8
#define TESTING_ROOM_FIRE              9
#define TESTING_ROOM_SMOKE             10
#define TESTING_ACTUAL_ROOM            11
#define TESTING_TEXTMSG                12
#define TESTING_DECORATION             13
#define TESTING_DECORATION_2           14


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

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


    /* Callback Prototypes: */

int  CVICALLBACK close_room1(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
```

```
int  CVICALLBACK close_room2(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK close_room3(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK close_room4(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK close_testing(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK display_testing(int panel. int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK fan_timer(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK fire_procedure(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK go_first(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK go_ground(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK go_second(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK power_function(int panel. int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK send_fan_speed1(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK send_fan_speed2(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK send_fan_speed3(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK send_fan_speed4(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK send_temp1(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK send_temp2(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK send_temp3(int panel. int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK send_temp4(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK show_history1(int panel. int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK show_history2(int panel. int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK show_history3(int panel. int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK show_history4(int panel. int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK show_room1(int panel. int control, int event, void *callbackData,
int eventData1, int eventData2);
```

```
int  CVICALLBACK show_room2(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK show_room3(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK show_room4(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK Shutdown(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK timer2_update(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK timer_update(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);


#ifdef __cplusplus
    }
#endif
```