ALEX MASUO KANEKO

Desenvolvimento de uma interface gráfica para supervisão de um sistema produtivo teleoperado

São Paulo
2008

ALEX MASUO KANEKO

Desenvolvimento de uma interface gráfica para supervisão de um sistema produtivo teleoperado

Monografia apresentada a Escola Politécnica da Universidade de São Paulo referente à disciplina PMR 2550 - Projeto de Conclusão do Curso II

Curso de Graduação: Engenharia Mecatrônica

Orientador: Prof. Dr. Paulo Eigi Miyagi

São Paulo
2008

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Paulo Eigi Miyagi, pela sua constante orientação e incentivo para o desenvolvimento deste trabalho.
Aos colegas de laboratório José, Marcos e Fabrício, pelo apoio e incentivo.
À empresa Siemens, em especial ao engenheiro Celso Souza, por compartilhar seu tempo e conhecimento, tão fundamentais neste trabalho.
À Escola Politecnica da USP, em especial ao Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, que institucionalmente viabilizaram este trabalho.

“Jovens, por favor, não se esqueçam do maior legado que os velhos imigrantes japoneses, assim como os outros imigrantes, deixaram: em qualquer situação, a ética e a moral, compreensão mútua entre nós para formarmos um país verdadeiramente humano, onde o coração vale mais do que muitas fortunas materiais.”

Kokei Uehara - 16 DE JUNHO DE 2008

028ª SESSÃO SOLENE EM HOMENAGEM AOS “CEM ANOS DA IMIGRAÇÃO JAPONESA NO BRASIL”

RESUMO

Com a evolução tecnológica e novos desafios de mercado, soluções distribuídas e dispersas geograficamente de sistemas de manufatura têm sido propostas. Este tipo de sistema procura explorar as potencialidades não só em relação a mercados específicos como também de recursos e cultura locais. Entretanto, a implementação disso depende de soluções efetivas para o problema de telecomando e monitoração remota de processos e equipamentos. O grupo de pesquisa em Mecatrônica da EPUSP está envolvido na rede KyaTera (do programa TIDIA da FAPESP) visando a implementação de weblabs para apoio à pesquisa de métodos de análise e projeto de sistemas produtivos dispersos e de soluções para a monitoração e diagnóstico remoto de processos e sistemas. Neste contexto, o presente trabalho apresenta o desenvolvimento de aplicativos computacionais que visam a implementação da interface com os operadores remotos de equipamentos. Desta forma, é disponibilizada a comunicação com os controladores locais e permite o gerenciamento dos equipamentos.

Este trabalho explora um conjunto de dispositivos e protocolos de comunicação amplamente utilizados em sistemas produtivos, como Controladores Programáveis, Servomotores, rede ASI e Profibus. Para o desenvolvimento das interfaces, foram utilizados conceitos e ferramentas, de modo a garantir eficiência para o controle do sistema produtivo. Assim, apresenta-se inicialmente os conceitos de sistemas produtivos, sistemas de controle, monitoração remota e ferramentas como *Sockets* e Visual Basic.

Palavras-chave: monitoração remota, sistemas produtivos dispersos, telecomando, interface.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SP – Sistema Produtivo

CP – Controlador Programável

CLP – Controlador Lógico Programável

SED – Sistema a Eventos Discretos

VB – Visual Basic

IDE – *Integrated Development Environment*

ASI – *Actuator Sensor Interface*

SUMÁRIO

1) Introdução

Os recursos computacionais têm avançado de modo significativo nos últimos anos e isso é observado pela sua intensa aplicação nas mais diversas áreas. O caso de sistemas produtivos (SPs), que são sistemas que realizam processos para a produção de bens ou serviços (VILLANI, 2003), não tem sido indiferente a esses avanços pois através do uso destes recursos computacionais procura-se assegurar maior qualidade e flexibilidade na produção de bens e serviços, o que envolve também maior complexidade das funções de controle nestes sistemas. Dentre as diferentes formas de organização de SPs, atualmente nota-se um crescente interesse pela distribuição física das partes o que envolve a necessidade de implementar soluções de telecomando e monitoração remota. Neste contexto uma estrutura básica deste tipo de sistema é representada na Figura 1. Este trabalho tem assim o objetivo de implementar uma interface gráfica de comunicação para um SP com a finalidade de monitorar remotamente e teleoperar os processos em um SP através da troca de informações entre os componentes do sistema.

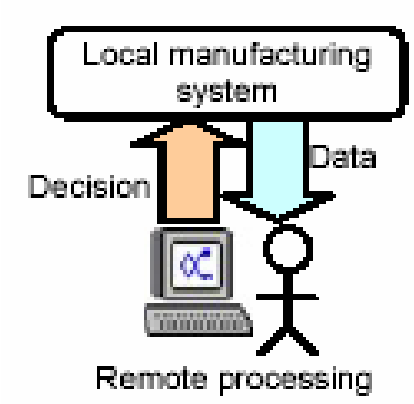

Figura 1 – Hierarquia de controle (VILLANI et al, 2006).

Interface neste trabalho é definida como o “ponto” em que há troca de sinais (informações/dados) entre dois dispositivos de *hardware*, entre um usuário e um programa ou sistema operacional, ou entre duas aplicações. No *hardware*, as interfaces são as conexões lógicas e físicas utilizadas, sendo considerada em geral um sinônimo de portas. A interface com o usuário se compõe dos meios

pelos quais um programa se comunica com o usuário, incluindo linhas de comando, menus, caixas de diálogos, recursos de ajuda *on line*, etc. Uma interface gráfica é uma interface que utiliza recursos gráficos (como imagens, setas, quadros, etc) para simplificar a visualização do estado do sistema e a sua operação de um sistema pelo usuário (DYSON, 1995).

Monitoração é o estudo e o acompanhamento (contínuo e sistemático) do comportamento de fenômenos, eventos e situações específicas, cujas condições deseja-se identificar, avaliar e comparar (ECOCÂMARA, 2008). A monitoração de um processo produtivo constitui uma importante tarefa na busca de maior eficiência e confiabilidade no funcionamento destes sistemas. Além disso, no caso de instalações industriais dispersas fisicamente um sistema de monitoração remota é fundamental já que é uma forma de disponibilizar recursos computacionais adequados e instalados em outros locais para os procedimentos complexos de diagnóstico e tratamento de falhas (MIYAGI et al, 2008).

A teleoperação é definida como a emissão de sinais de comando visando o controle de processos por um operador através de um dispositivo localizado remotamente. Desenvolvida entre outras coisas para a manipulação de materiais radioativos, a teleoperação permite que um operador realize funções à distância através de dados visuais, sonoros ou táteis. Com a introdução da tecnologia de teleoperação, tem-se desenvolvido interfaces capazes de prover uma interação satisfatória entre homem e máquina, permitindo que serviços de grande destreza sejam realizados remotamente (ÁLVARES & SOUZA, 2001).

SPs, já definidos anteriormente, não se limitam apenas a sistemas de manufatura, isto é, envolvem também sistemas prediais, sistemas de transporte, etc (VILLANI, 2003). Quando se trata de um SP distribuído, entende-se que os recursos que realizam o processo estão distribuídos em diferentes locais físicos, e a interação dos sinais de comando e monitoração ocorre através de redes de comunicação (MATSUSAKI, 2004).

O SP considerado como estudo de caso deste trabalho é o “sistema flexível de montagem” da empresa Festo, instalado no Departamento de Engenharia Mecatrônica da EPUSP e que executa um processo de montagem

automática de diferentes produtos. Cada produto é composto por quatro tipos de peças: uma base (que tem três cores possíveis: preta, prateada e rosa), um pino (que tem 2 cores possíveis: preto ou cinza); uma mola e uma tampa. De acordo com a cor da base, os produtos montados assumem diferentes composições, como pode ser observado na Figura 2.

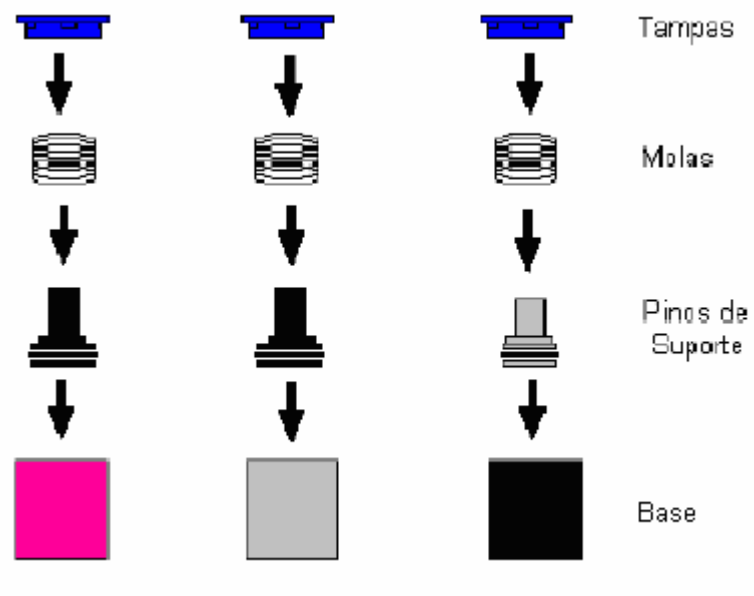

Figura 2 – Montagem dos diferentes produtos pelo SP.

O SP na versão atualmente instalada e em operação é um conjunto de 4 estações de trabalho, cada qual capaz de ser operada individualmente ou, no contexto de um sistema integrado e automatizado, no qual cada estação possui certa autonomia na execução de suas tarefas. As estações de trabalho são as seguintes: estação de distribuição, estação de testes, estação de transporte e estação de montagem.

A estação de distribuição armazena as bases dos produtos a serem montados e os encaminha ao processo de produção (estação de testes) de acordo com a demanda. Já a estação de testes é responsável pela identificação da cor (prateada, rosa ou preta) e teste da altura das bases. Esta estação é também responsável por descartar peças rejeitadas neste teste. A estação de transporte conta com uma série de esteiras de transporte e quatro carros (pallets). Esta estação de transporte tem definido posições distintas para a

parada dos carros, sendo um deles destinado à estação de testes e outro à estação de montagem. A estação de transporte destina-se a transportar as bases identificadas e testadas da estação de testes para a estação de montagem, onde a montagem dos produtos é finalizada. Os produtos montados são então transportados pelo sistema de transporte para uma outra localidade distinta que seria uma estação de armazenagem que ainda não foi instalada. Um esquema ilustrativo do SP é mostrado na Figura 3:

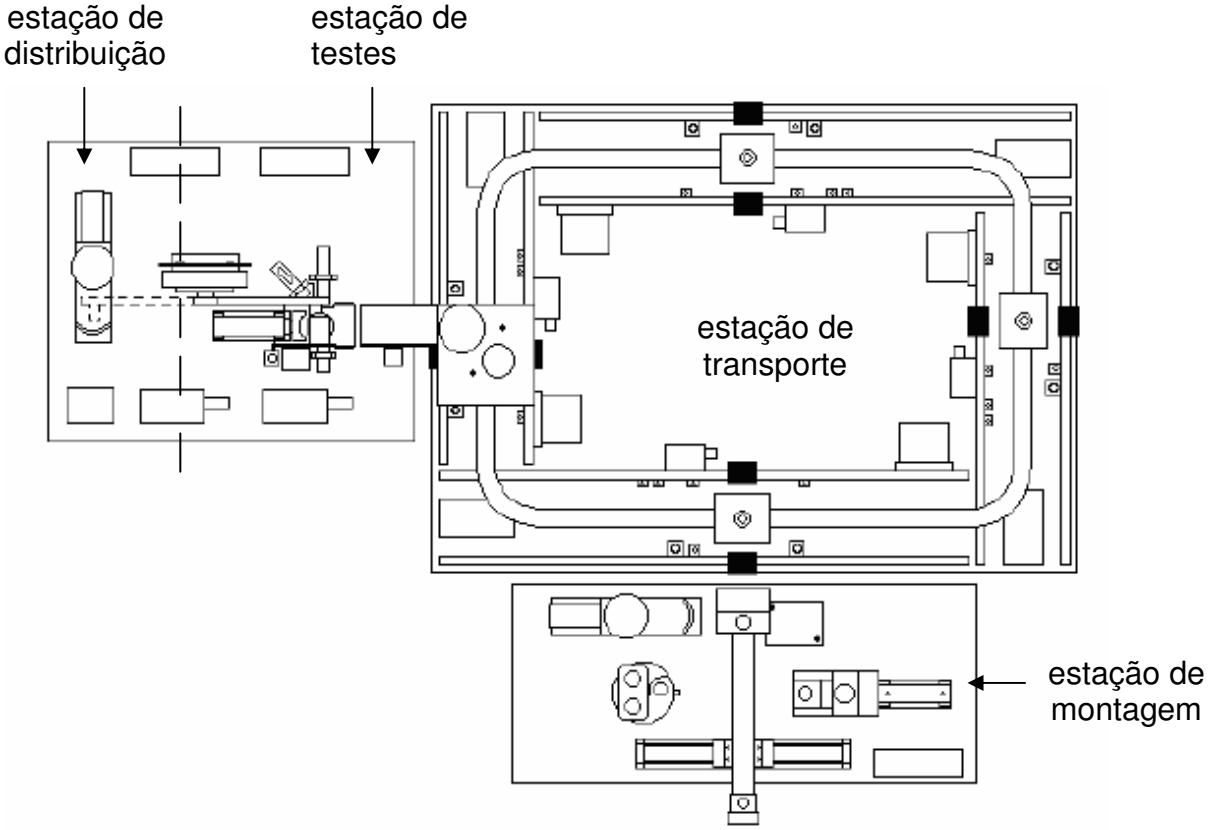

Figura 3 – Esquema ilustrativo do SP

Entre as facilidades operacionais do SP encontram-se:

a) Estação de distribuição e estação de testes: possuem dispositivos para manipulação, testes de controle de qualidade através de sensores, dispositivos de aquisição e avaliação de sinais digitais e analógicos, assim como um sistema de controle para cada estação, com controladores programáveis (CPs),

conectados através de um sistema de comunicação denominado PROFIBUS (*Process Field Bus*);

b) Estação de montagem: além de plataformas de fixação e dispositivos para armazenar as peças a serem montadas, possui um manipulador de 3 graus de liberdade para a montagem destas peças, sendo que dois dos eixos são acionados por servo-motores (eixos x e y, no plano horizontal) e o terceiro eixo por motor CC (eixo z, no plano vertical). Este manipulador é dotado ainda de dispositivo de garra dedicado. O controle é realizado por CP e inclui comunicação via PROFIBUS.

c) Estação de transporte: possui um conjunto de esteiras, dispositivos pneumáticos para fixação e movimentação de carros (pallets) e indexadores, que definem os pontos de parada dos carros, além de futuras expansões.

Apesar de o SP considerado ser composto pelas quatro estações de trabalho descritas, para este trabalho o foco está no controle das estações de transporte e de montagem.

1.1) Organização do texto

No capítulo 1 foi feita uma introdução do trabalho, apresentando as motivações e o sistema produtivo considerado. No capítulo 2 é feita uma descrição dos principais conceitos e ferramentas envolvidas no trabalho. O capítulo 3 descreve a estrutura geral de controle do sistema produtivo considerado. O capítulo 4 discute os modos de operação e uso das interfaces e no capítulo 5 apresenta-se como a implementação e os testes das interfaces gráficas foram realizados. As conclusões do trabalho são apresentadas no capítulo 6, seguidas das referências bibliográficas. Por último, seguem os anexos.

2) Conceitos e ferramentas

A seguir são apresentados os principais conceitos, teorias, tecnologias e ferramentas consideradas para o desenvolvimento do presente trabalho.

Do ponto de vista da teoria, foram considerados inicialmente conceitos como controle de sistemas produtivos (SPs), sistemas de controle de sistema a eventos discretos, monitoração remota, manufatura dispersa, telediagnóstico de máquinas, controle supervisório, Profibus, *Actuator Sensor Interface* (ASI) e controladores programáveis (CPs).

No que se refere à implementação, foram utilizadas como ferramentas os softwares STEP7 e Parameterize FM354, Visual Basic, Prodave e *Sockets*.

2.1) Controle de SPs

Controle pode ser definido como a aplicação de uma ação pré-planejada para que aquilo que se considera como objeto de controle atinja certos objetivos (MIYAGI, 1996). O controle de SPs pode então ser definido como um sistema que garante a eficiência, eficácia e integridade na execução dos processos produtivos especificados (VILLANI, 2003).

2.2) Sistemas de Controle de sistemas a eventos discretos

Um sistema de controle de sistema a eventos discretos (SED) envolve um conjunto de dispositivos que, baseado num procedimento pré-estabelecido ou numa lógica fixa que estabelece um procedimento, executa ordenadamente cada estágio do controle.

Este tipo de sistema é o que melhor representa a implementação de controle automático qualitativo (o conteúdo dos comandos de controle possuem informações, que são classificadas dentro de um conjunto finito de valores) e sua principal característica é que o objeto de controle trabalha com estados e eventos discretos.

No controle de SED, as operações fundamentais (lógicas, aritméticas e temporizações) são realizadas por dispositivos tais como operadores lógicos, operadores aritméticos e temporizadores, que possuem sinais de entrada e/ou saída que podem assumir valores discretos (MIYAGI, 1996).

2.3) Monitoração remota

Prevenir, deter, detectar e responder são as ações que definem a importância de uma monitoração remota em um sistema que envolve uma rede de comunicação (ROCHA, 2003).

Segundo SOUZA (2002), as redes de comunicação estão presentes na sociedade, interligando desde computadores pessoais em uma casa até os dispositivos computacionais de uma empresa. Por facilitar o desenvolvimento de trabalhos e permitir o acesso a informações *on line* na internet, as redes de comunicação são cada vez mais objeto de novos serviços oferecidos pelas empresas e instituições, por isso torna-se vital mantê-las sempre disponíveis e com bom desempenho.

A monitoração remota é empregada em diversos casos, desde o apoio para as operações de máquinas ferramentas até em serviços de bancos e hospitais. Neste último, a monitoração remota está sendo usada para o acompanhamento *on line* de pacientes através da internet, transmitindo dados com alta qualidade e confiabilidade (Figura 4) (CHRIST et al., 2006).

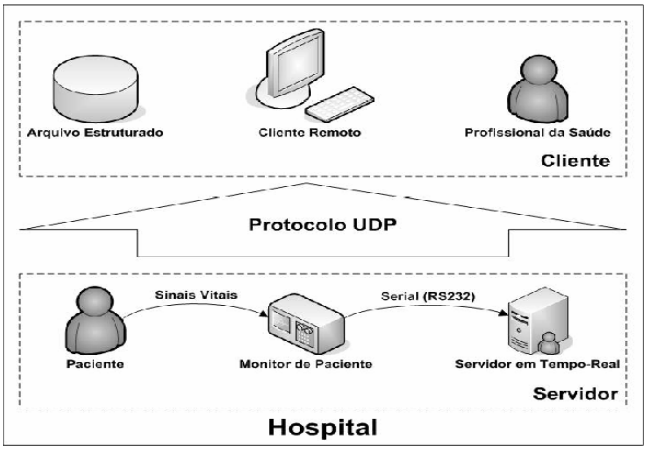

Figura 4: Exemplo de monitoração remota em um hospital (CHRIST et al., 2006)

2.4) Manufatura dispersa

Por manufatura dispersa (ou distribuída) entende-se como uma combinação de conceitos emergentes voltados para o gerenciamento e controle distribuído da manufatura, que surgiu como tentativa de tratar e explorar a autonomia de componentes do sistema, visando torná-lo mais competitivo em um ambiente com facilidades de comunicação e diversidade de recursos distribuídos geograficamente (GOULART, 2000).

2.5) Telediagnóstico de máquinas

Entende-se por telediagnóstico o processamento de dados e/ou imagens para análise de processos em geral relacionados à ocorrência de situações anormais (FERREIRA, 2000). Esta tecnologia está sendo amplamente empregada na medicina e está se expandindo para as máquinas ferramentas, permitindo otimizar ações de manutenção, tanto em ações corretivas (após a falha), quanto em ações preventivas (antes da falha).

2.6) Controle supervisório

O termo “supervisório” geralmente está associado à estrutura de tomada de decisões nas indústrias em relação ao acompanhamento *on line* dos processos de produção no "chão de fábrica" (ALLCOMNET, 2006) ou no nível da célula de manufatura. Representa um nível de controle superior (logo acima do chão de fábrica) que, por exemplo, gerencia as tarefas de um conjunto de controladores programáveis (CPs), que por sua vez atuam diretamente nos equipamentos do processo (Figura 5).

No contexto das indústrias de processo, o controle supervisório denota um sistema que gerencia as atividades de operações integradas para alcançar certos objetivos organizacionais e de produção. Em muitas aplicações, esse gerenciamento envolve a definição dos valores de regulação por

retroalimentação, em outras ele é projetado para indicar o parâmetro de operação ótimo ou adaptativo.

No caso da manufatura discreta, o controle supervisório envolve a direção e a coordenação das atividades (tarefas) dos equipamentos em uma célula ou sistema de manufatura, tais como um grupo de máquinas interconectadas por um sistema de manipulação de material. Nesse caso, o controle supervisório considera objetivos tais como: minimizar custos de peças ou produtos pela manutenção de condições ótimas de operação, maximizar a utilização de máquinas através de seqüenciamento eficiente de tarefas, minimizar custos ferramentais pelo acompanhamento da vida de ferramentas e pelo seqüenciamento de trocas de ferramentas (MARTINS FILHO, 2006).

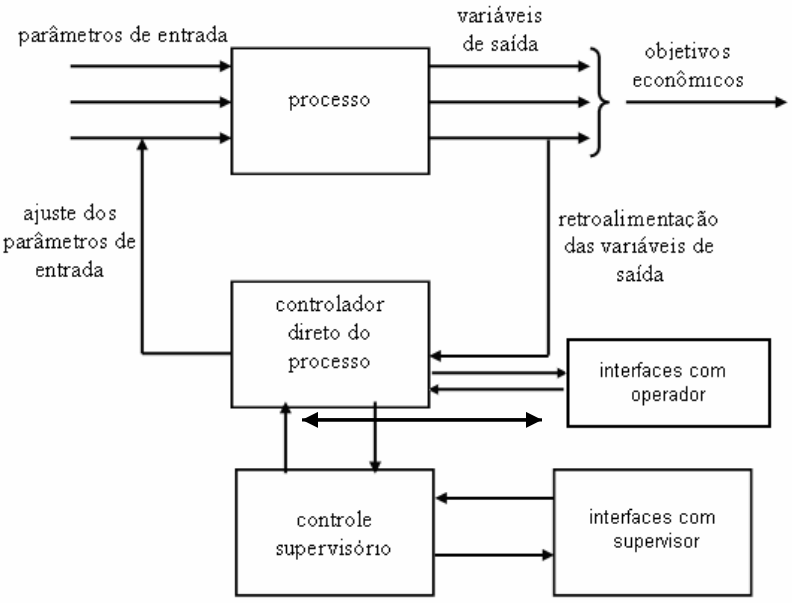

Figura 5: Exemplo de uma planta de um controle supervisório (adaptado de MARTINS FILHO, 2006)

2.7) Profibus

O SP considerado como estudo de caso, citado no Capítulo 1, adota o Profibus para a rede de comunicação local entre os CPs que formam o sistema de controle da linha de produção e o supervisor local.

O Profibus começou como um projeto apoiado por autoridades públicas, que iniciou em 1987 na Alemanha. Dentro do contexto deste projeto, 21 companhias e institutos uniram forças e criaram um projeto estratégico *fieldbus*. O objetivo era a realização e manutenção de um barramento de campo bitserial, sendo o requisito básico a padronização da interface de dispositivo de campo. Por esta razão, os membros das companhias do ZVEI (Associação Central da Indústria Elétrica Alemã) concordaram em adotar um conceito técnico para manufatura e automação de processos (PROFIBUS, 2006).

Um primeiro passo foi a especificação do protocolo de comunicação chamada de Profibus FMS (especificação de mensagens *fieldbus*), que foi "costurado" para atender as exigências de tarefas de comunicação.

Mais adiante, em 1993, tem-se a conclusão da especificação do Profibus DP (Periferia Descentralizada).

Baseado nestes dois protocolos de comunicação e, acoplado com o desenvolvimento de numerosos perfis de aplicações, o Profibus começou seu avanço inicialmente na automação da manufatura, e desde 1995, na automação de processos. Hoje, o Profibus é um barramento relativamente conhecido no mercado mundial.

O Profibus é um padrão de rede de comunicação de campo, aberto e independente de fornecedores, onde a interface entre dispositivos permite uma ampla aplicação em processos, manufatura e automação predial.

Ele também pode utilizar como meio de transmissão qualquer um dos seguintes padrões: RS-485, IEC 61158-2 ou fibra ótica.

2.7.1) Profibus-DP - Periferia Descentralizada

O Profibus especifica as características técnicas e funcionais de um sistema de comunicação industrial, através do qual dispositivos digitais podem se interconectar, do nível de campo como os CPs até o nível de sensores, atuadores, válvulas, etc (Figura 6).

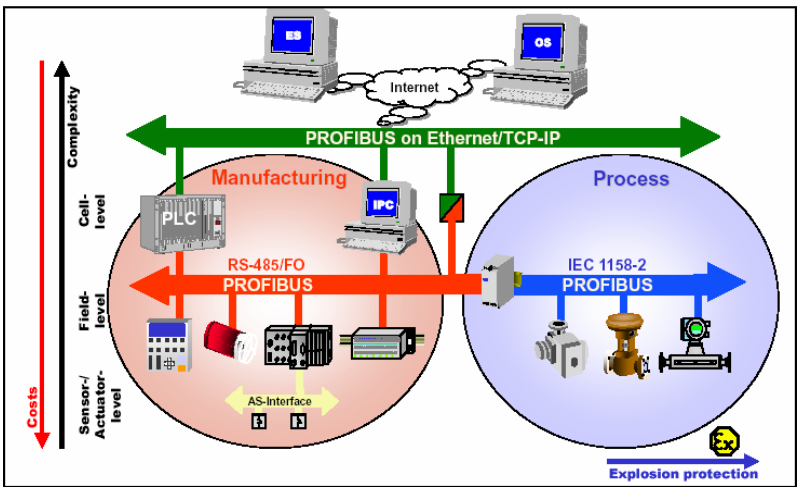

Figura 6 - Comunicação Industrial (PROFIBUS, 2006).

Trata-se de um sistema multi-mestre que permite a operação conjunta de diversos sistemas de automação, engenharia ou visualização, com seus respectivos dispositivos periféricos (por ex. I/O's). Ele diferencia seus dispositivos entre mestres e escravos.

Dispositivos mestres determinam a comunicação de dados no barramento. Um mestre pode enviar mensagens, sem uma requisição externa, sempre que possuir o direito de acesso ao barramento. Os mestres também são chamados de estações ativas no protocolo Profibus.

Os dispositivos escravos são dispositivos remotos (de periferia), tais como módulos de I/O, válvulas e transdutores. Eles não têm direito de acesso ao barramento e só podem enviar mensagens ao mestre ou reconhecer mensagens recebidas quando solicitados. Os escravos também são chamados estações passivas.

O Profibus-DP, otimizado para alta velocidade de transmissão e conexão de baixo custo, foi projetado especialmente para a comunicação entre sistemas de controle de automação e seus respectivos I/Os distribuídos no nível de dispositivo.

2.7.2) Meio de transmissão RS-485

O padrão RS-485 é a tecnologia de transmissão mais freqüentemente encontrada no Profibus. Sua aplicação inclui todas as áreas nas quais uma alta taxa de transmissão aliada à uma instalação relativamente simples e barata são necessárias. Um par trançado de fio de cobre blindado (*shieldado*) é o suficiente neste caso.

A tecnologia de transmissão RS-485 é relativamente fácil de manusear (Tabela 1). O uso de par trançado não requer nenhum conhecimento ou habilidade especial. A topologia por sua vez permite a adição e remoção de estações (outros CPs), bem como uma implantação em etapas, sem afetar a instalação existente. Expansões futuras, portanto, podem ser implementadas sem afetar as estações já em operação.

Taxas de transmissão entre 9.6 kbit/s e 12 Mbit/s podem ser selecionadas, porém uma única taxa de transmissão deve ser utilizada para todos dispositivos no barramento, quando o sistema é inicializado.

Na instalação do RS-485 todos os dispositivos são conectados a uma estrutura de tipo barramento linear que pode ser composto por vários segmentos. Até 32 estações (mestres ou escravos) podem ser conectados a um único segmento. O barramento é terminado por um "terminador ativo" do barramento no início e fim de cada segmento. Para assegurar uma operação livre de erros, ambas as terminações do barramento devem estar sempre ativas. Normalmente estes terminadores encontram-se nos próprios conectores de barramento ou nos dispositivos de campo, acessíveis através de uma *dip-switch*. No caso em que mais que 32 estações necessitem ser conectadas ou no caso em que a distância total entre as estações ultrapasse um determinado limite, devem ser utilizados repetidores (*repeaters*) para se interconectar diferentes segmentos do barramento.

O comprimento máximo do cabo depende da velocidade de transmissão (Tabela 2). As especificações de comprimento de cabo na Tabela 2, são baseadas em cabo com os seguintes parâmetros:

- Impedância: 135 a 165 ohms
- Capacidade: < 30 pF/m
- Resistência: 110 ohms/km
- Medida do cabo: 0.64mm
- Área do condutor: > 0.34mm²

Tabela 1 - Características básicas do RS-485

Mídia	Cabo tipo par trançado blindado. A blindagem pode ser omitida, dependendo das condições eletromagnéticas do ambiente (EMC).
Número de Estações	32 estações em cada segmento sem repetidores. Com repetidores pode ser estendida até 126 estações.
Conectores	Preferencialmente DB-9 para IP20. M12, Han-Brid ou tipo Híbrido para IP65/67.

Tabela 2 - Distâncias baseadas em velocidade de transmissão

Baud rate (Kbit/s)	9.6	19.2	93.75	187.5	500	1500	12000
Distância/segmento (m)	1200	1200	1200	1000	400	200	100

Os cabos para o Profibus são oferecidos por vários fabricantes. Durante a instalação, deve-se observar atentamente a polaridade dos sinais de dados. O uso da blindagem é essencial para se assegurar alta imunidade contra interferências eletromagnéticas. A blindagem por sua vez deve ser conectada ao sistema de aterramento em ambos os lados através de bornes de aterramento adequados. Adicionalmente recomenda-se que os cabos de comunicação sejam mantidos separados dos cabos de maior voltagem. O uso de cabos de derivação deve ser evitado para taxas de transmissão acima de 1,5Mbits/s. Os conectores disponíveis no mercado permitem que o cabo do barramento “entre/saia” diretamente no conector, permitindo assim que um dispositivo seja conectado/desconectado da rede sem interromper a comunicação.

Nota-se que quando problemas ocorrem em uma rede Profibus, cerca de 90% dos casos são provocados por incorreta ligação e/ou instalação. Estes problemas podem ser facilmente solucionados com o uso de equipamentos de teste, os quais detectam falhas nas conexões.

Para a conexão em locais com grau de proteção IP20, utilizam-se conectores tipo DB9 (9 pinos). A definição da pinagem e esquema de ligação são mostrados na Figura 7 (PROFIBUS, 2006).

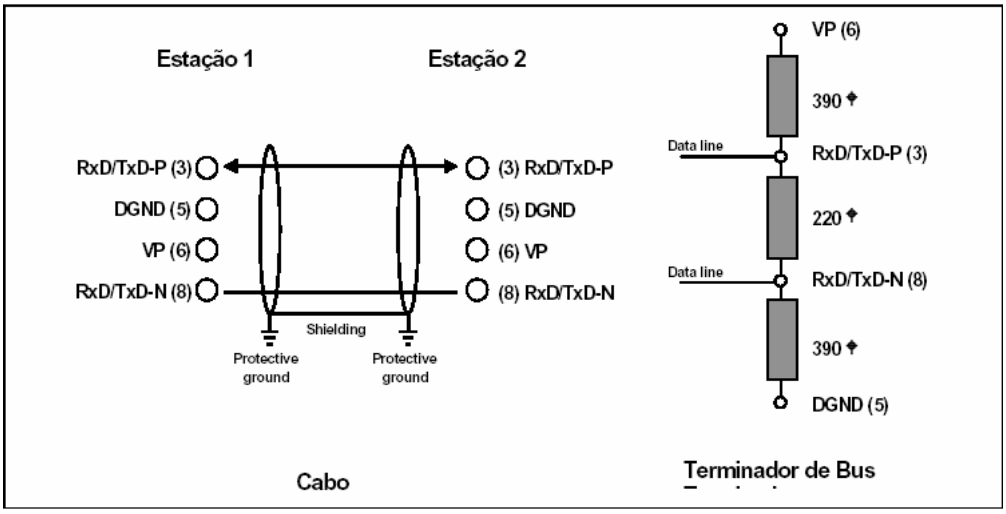

Figura 7 - Ligação e terminação para o RS-485 (PROFIBUS, 2006).

2.8) CP

Para o controle da estação de montagem do SP apresentado no Capítulo 1, é necessário o estudo de seu CP, que no caso, é um Controlador Programável Siemens SIMATIC S7-300 com uma CPU 315-2 DP.

Um Controlador Lógico Programável, ou Controlador Programável conhecido também por suas siglas CLP ou CP no Brasil e pela sigla de expressão inglesa *Programmable Logic Controller* (PLC), pode ser visto como um computador especializado para ambiente industrial, baseado num microprocessador que desempenha funções de controle de diversos tipos e níveis de complexidade. O controlador permite o desenvolvimento de sistemas caracterizados por eventos discretos, ou seja, com processos em que as variáveis assumem valores zero ou um (ou variáveis ditas digitais, ou seja, que só assumem valores dentro de um conjunto finito). Podem ainda lidar com variáveis analógicas definidas por intervalos de valores de corrente ou tensão elétrica.

Os CPs estão difundidos nas áreas de controle de processos ou de automação industrial. No primeiro caso a aplicação se dá nas indústrias do tipo contínuo, envolvendo o processamento de fluidos e semi-fluidos, misturas (fluidos e sólidos), no outro caso a aplicação se dá nas áreas relacionadas com o processamento discreto em lotes ou batelada, por exemplo na linha de montagem de automóveis. Num sistema típico, a informação dos sensores é concentrada no CP que de acordo com o programa em memória define o estado dos pontos de saída conectados a atuadores.

Os CPs têm capacidade de comunicação de dados via canais seriais. Com isto podem ser supervisionados por computadores formando sistemas de controle integrados. Esta supervisão envolve *softwares* específicos para monitorar os dados de um conjunto de CPs, processar estes dados e auxiliar na tomada de decisões pelo "supervisor" e enviar os comandos/ajustes devidos para os CPs. Isto evidentemente envolve a disponibilidade de uma rede de comunicação entre o computador (com o software de supervisão) e os CPs. Redes de comunicação abertas como PROFIBUS-DP são de uso muito comum com CPs permitindo aplicações complexas na indústria automobilística, siderurgica, de papel e celulose, e outras (WIKIPEDIA, 2006 (a)).

2.8.1) Controlador programável S7-300

O SIMATIC S7-300 é um controlador modular amplamente utilizado em aplicações industriais centralizadas ou distribuídas de pequeno a médio porte.

A Figura 8 apresenta a estrutura do SIMATIC S7-300, que é composta por:

1 - Fonte de energia (opcional)
2 - Bateria de reserva (CPU 313 e acima)
3 - Conector 24 V DC
4 - Chave de operação.
5 - Status e falhas (leds)

6 - Cartão de memória (CPU 313 e acima)

7 - MPI (Interface multi-ponto)

8 - Conector frontal

9 - Porta da frente.

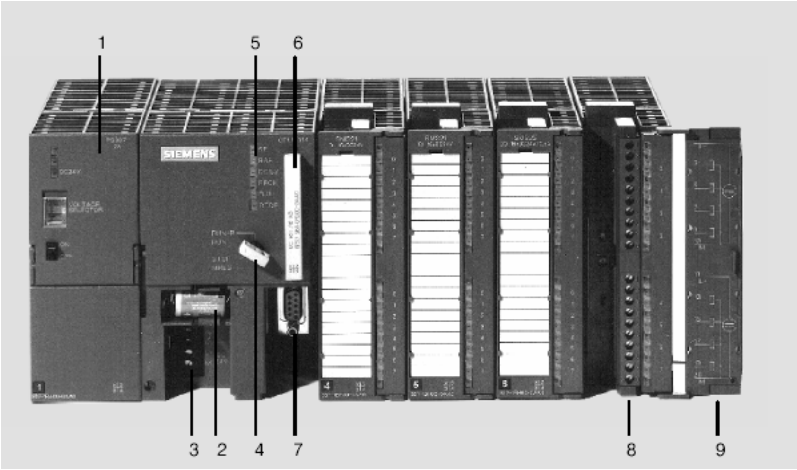

Figura 8 - SIMATIC S7-300 design (PINHEIRO, 2006).

Este modelo tem integrado a si uma porta MPI e uma segunda interface de comunicação integrada Profibus (PINHEIRO, 2006).

Ele possui uma CPU que possibilita tempos de ciclo de até 1µs por instrução binária (Tabela 3).

Tabela 3 – Especificação do CPU 315-2 DP.

	CPU 315-2 DP
Memória principal	64 Kbyte
Tempos de processamento • Operações binárias • Operações com palavras	 0,3 µs a 0,62 µs 1µs
Faixas de endereçamento • Área total de endereçamento de E/S • Área de Imagem de processo E/S • Canais digitais totais • Canais analógicos totais	 1/1 kbyte 128/128 byte máx. 8192 (1024 centrais) máx. 512(máx. 256 entradas ou 128 saídas centrais)

Interfaces Profibus DP	
• Número de segmentos DP integrado/CP342-5	1 / 1
• No. estações escravas DP por CPU (interfaces integradas / CP 342-5)	64/64
• Velocidade de transmissão	12 Mbit/s

Este CP dispõe de diversos módulos de expansão que permite a adaptação da configuração para diferentes tipos de aplicação (Tabela 4):

Tabela 4 – Módulos de expansão para o CLP SIMATIC S7-300.

Módulos de I/O	- Digitais
	- Analógicos
Módulos de Comunicação	-Profibus DP
	-Ethernet
	- AS-interface
	- Serial Ponto-a-Ponto
	- Modbus
Módulos de Função	- Contadores rápidos
	- Controle de posição
	- Controle de motor de passo
	- Controle em malha fechada
	- Saídas de pulso rápida

Um total de até 32 módulos de expansão podem ser utilizados em uma configuração centralizada.

A programação deste CLP é feita por meio do software de apoio STEP 7 (Figura 9), que pode ser realizada através de linguagens como ladder, diagrama de blocos ou lista de instruções, complementadas por três ferramentas:

- S7-GRAPH para a programação gráfica de controles seqüenciais.
- S7-SCL, para uma linguagem de alto nível que visa facilitar o tratamento de tarefas mais complexas.
-S7-PLCSIM para a simulação "*offline*" da sua solução de automação.

Um programa pode ser totalmente criado em uma dessas linguagens, ou então pela combinação delas, cabendo ao usuário decidir a qualquer momento do programa qual dessas linguagens se deseja usar (Figura 10).

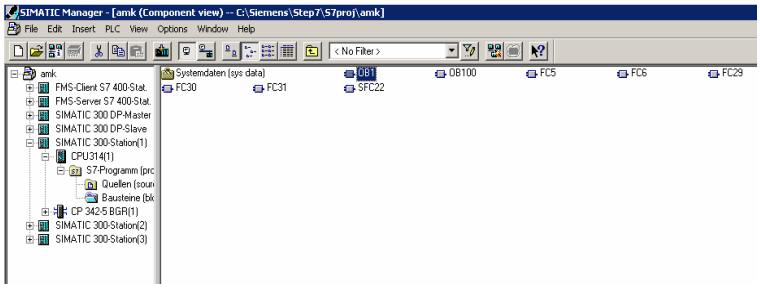

Figura 9 – Janela principal do STEP 7

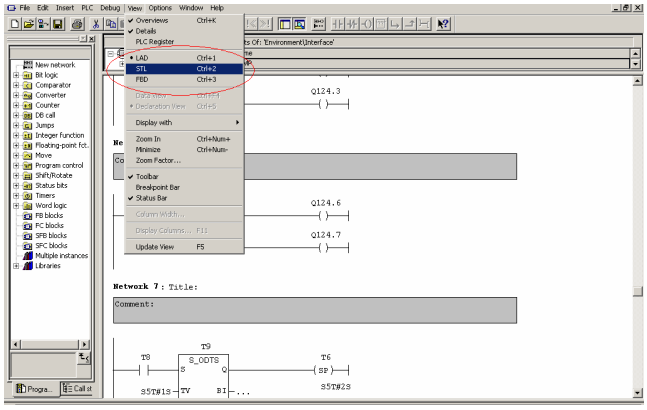

Figura 10 – Janela do STEP7 com as diferentes possibilidades de linguagem

2.9) ASI

O SP apresentado no capítulo 1 é composto por diversos atuadores e sensores. Desta maneira, para organizar como estes dispositivos devem ser conectados aos CPs.

A *Actuator Sensor Interface* (ASI) é uma solução de rede de sensores e atuadores. Normalmente, os sinais dos sensores e atuadores dos processos industriais são transmitidos através de um grande número de cabos. O sistema ASI permite a simplificação deste sistema de fiação e ligação, substituindo o então sistema de cabos, por apenas um par de fios, que podem ser usados por todos os sensores e atuadores. Este par de fios é responsável pela alimentação (de energia) dos sensores/atuadores e pela transmissão dos dados binários de

entrada e saída. A Figura 11 ilustra a posição da ASI na hierarquia na comunicação industrial.

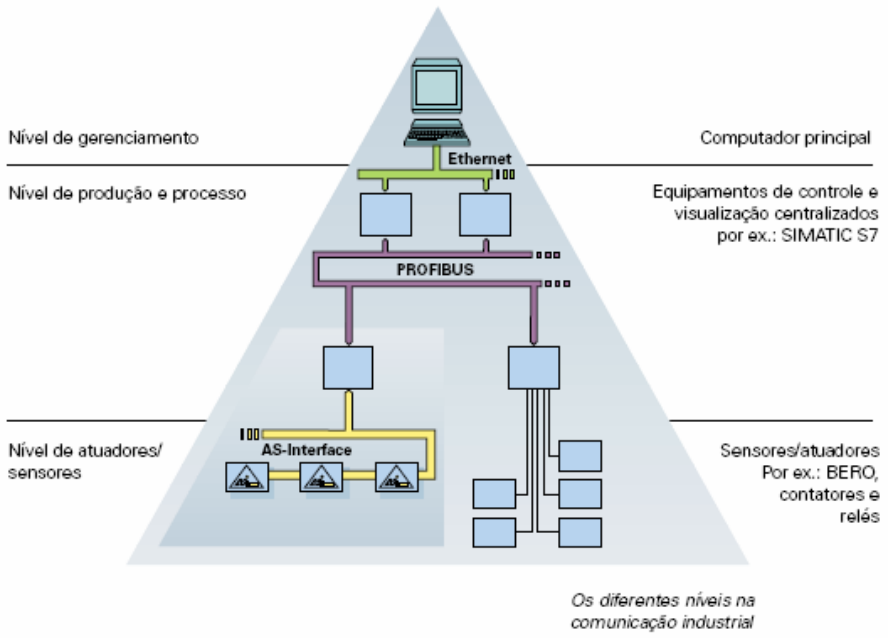

Figura 11 - Hierarquia da comunicação industrial (SIEMENS (a), 2008)

A ASI foi concebida para complementar os demais sistemas e tornar mais simples e rápida a conexão dos sensores e atuadores com os seus respectivos controladores. O sistema baseia-se numa comunicação mestre-escravo, cujo mestre é responsável pelo direcionamento das "perguntas" e tratamento das "respostas" dos escravos. O mestre pode gerenciar até trinta e um escravos. A comunicação entre o mestre e os escravos é feita serialmente, através de um par de fios não trançados e nem blindados. Inicialmente, o mestre "fala" com o primeiro escravo, atualiza as saídas do mesmo (se existirem) e pergunta o estado binário das entradas deste escravo. Imediatamente o escravo responde e, após um pequeno intervalo, o mestre "fala" com o próximo escravo. Após o último escravo, o ciclo se completa e o mestre começa a conversar novamente com o primeiro escravo. O ciclo de varredura completo tem duração de até 5 ms (contendo 31 escravos na rede). Um escravo ASI pode possuir no máximo 4 entradas digitais e no máximo 4 saídas digitais (ALTUS, 2008).

Neste projeto, o mestre ASI é implementado por um módulo CP342-2 do fabricante Siemens. Para a comunicação entre o mestre e os escravos, são usados quatro módulos I/O do fabricante Festo (Figura 12), que permite a ligação física dos sensores e atuadores na rede.

Figura 12 – Módulo ASI utilizado no projeto (SIEMENS (a), 2008).

A rede ASI permite o uso de vários tipos de topologias, permitindo ainda que a qualquer momento possa se iniciar uma nova derivação, possibilitando a inclusão de novos sensores e atuadores. Cada usuário pode escolher sua topologia, conforme sua necessidade e disposição física dos elementos no campo. O cabo da rede não necessita de resistor de terminação, sua única limitação está relacionada com o comprimento do fio, que deve possuir o máximo de 100 m de comprimento. Caso necessário, o cabo pode ter um acréscimo de 200 m com a utilização de repetidores (*boosters*), ficando, assim, com um comprimento total de 300 m (ALTUS, 2008).

O cabo amarelo e perfilado (Figura 13), padrão da AS-Interface, tornou-se um tipo de marca registrada. Ele possui uma seção geometricamente determinada e transmite ao mesmo tempo dados e energia auxiliar para os sensores. Para os atuadores é em geral, necessária uma tensão auxiliar alimentada adicionalmente (24VCC).

Para se poder utilizar a mesma técnica de instalação para os atuadores, foram especificados cabos com as mesmas características, mas de outra cor. Desta forma, o cabo para a energia auxiliar 24VCC é um cabo perfilado preto (Figura 13).

Para aplicações com exigências maiores podem se utilizar cabos com outras composições químicas como: TPE perfilado (elastômetro termoplástico) ou PUR perfilado (poliuretano). Como condutor de transmissão podem ser utilizados também cabos redondos com sistema de condução duplo. Uma blindagem do condutor não é necessária em função da técnica de transmissão empregada (Siemens, 2008 (a)).

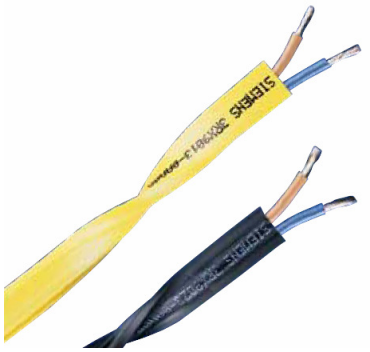

Figura 13 – Cabos ASI (SIEMENS (a), 2008).

2.10) Prodave

Para a comunicação entre um computador e um CP, é preciso um conjunto de bibliotecas que permitem a troca de dados entre os dispositivos. Neste contexto, o estudo do Prodave se torna necessário.

O Prodave é um aplicativo, criado pela empresa Siemens, composto por um conjunto de funções em formato DLL (sendo as principais “komfort.dll” e “w95_s7.dll”) ou LIB os quais podem ser combinadas e usadas em aplicações Windows. Por meio dessas funções a troca de dados (entradas e saídas) entre CP e computador é estabelecida usando a interface MPI ou Profibus do CP (SIEMENS (b), 2005).

Algumas das funções usadas nesse trabalho são listadas a seguir:

- load_tool: Esta função inicializa o adaptador (no caso, um adaptador que permite a conexão física entre o cabo Profibus e a entrada do CP), verifica se o *driver* (da comunicação) está carregado, inicializa os endereços parametrizados (endereços a serem usados no programa) e altera a interface (neste caso, uma

placa de comunicaçao Profibus entre computador e CP) selecionada para “ativo”.

- unload_tool: Esta função finaliza as conexões abertas entre computador e CP.
- m_field_read: Esta função lê flags do CP e armazena os valores em uma variável (definida pelo usuário).
- m_field_write: Esta função escreve flags no CP em endereços determinados pelo usuário.

Para ilustrar um exemplo do uso dessas bibliotecas, ao acessar no menu o comando “write“ da tela principal do Prodave (Figura 14), tem-se no lado esquerdo da janela opções para o tipo de dados (entrada, *flag*, contador, *flag* especial, etc) a serem enviados ao CP. No lado direito da janela, em “Datablock”, “from number” e “amount”, é feita a escolha da posição (endereço) da memória do CP em que se deseja escrever. Cada posição da memória do CP é responsável por determinadas funções, e de acordo com o valor de um determinado endereço da memória, é gerada uma diferente ação da respectiva estação do SP ilustrado no capítulo 1. Essas ações também podem ser consideradas como “passos” para o processo de produção da respectiva estação do SP, e através de uma lógica adequada, é possível transformar passos isolados em um processo completo de manufatura, como no caso deste trabalho.

Em “value”, é inserido o dado a ser enviado ao CP. Estes dados podem ser escritos em hexadecimal, decimal ou em binário, de acordo com a preferência do usuário.

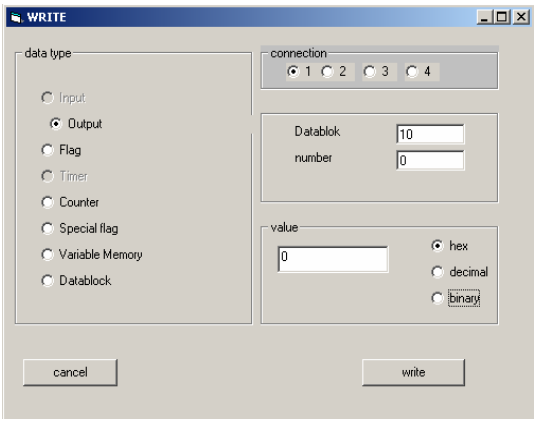

Figura 14 – Janela "Write" do Prodave

Outra importante opção do Prodave é a função "read" (Figura 15), que mostra os valores dos dados em qualquer posição da memória do CP. Para fazer a leitura, o procedimento é análogo ao da função "write", porém há ainda o recurso "cycle read" que faz a leitura automática dos dados a cada instante de tempo determinado.

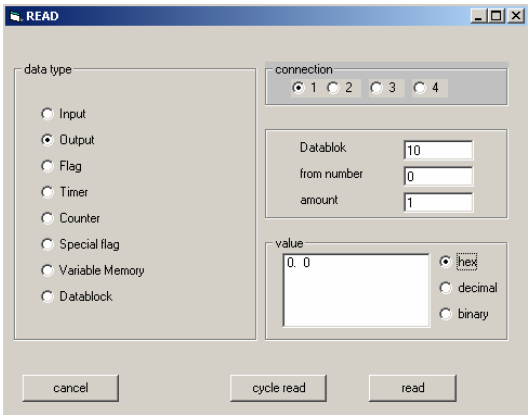

Figura 15 – Janela "Read" do Prodave.

É importante destacar que os endereços de memória a serem escritos devem ser previamente conhecidos, por meio de uma tabela fornecida pelo fabricante do CP ou por testes, relacionando as saídas do CP com as suas entradas. Os endereços utilizados neste trabalho serão exibidos nos próximos capítulos deste relatório.

2.11) Visual Basic

Para a implementação das interfaces gráficas nos computadores local e remoto é preciso uma linguagem de programação que facilite a construção de recursos gráficos e que tenha alta confiabilidade no aspecto de integração com bibliotecas para comunicação com os CPs, e também na integração remota entre computadores.

O Visual Basic (VB) é uma linguagem de programação produzida pela empresa Microsoft. A linguagem é dirigida por eventos, e possui também um ambiente de desenvolvimento integrado (IDE) (Figura 16) totalmente gráfico, facilitanto a construção da interface das aplicações, daí o nome "Visual". Em suas primeiras versões, o VB não permitia acesso a bancos de dados, sendo portanto voltado apenas para iniciantes, mas devido ao sucesso entre as empresas a linguagem logo adotou recursos mais sofisticados permitindo fácil acesso a bases de dados. Mais tarde foi adicionada também a possibilidade de criação de controles como o ActiveX, fundamentais para este projeto (BRASILIA VIRTUAL, 208).

Com o passar do tempo, escrever programas passou a ser cada vez mais difícil, especialmente programas que exigem interface gráfica. Entretanto, alguns programadores perceberam que muitas coisas que eram difíceis de serem feitas, como construir janelas, menus ou botões, podiam ser feitas sempre da mesma forma. Estes programadores, que já tinham o hábito de colecionar sub-rotinas de utilização geral, passaram a encapsular algumas destas rotinas em uma espécie de "objeto" pronto para ser usado. A idéia final, que deu origem ao IDE, foi a percepção de que vários destes objetos podiam simplesmente ser desenhados na tela como se desenha um retângulo ou outra figura qualquer (ALMEIDA, 1999). O IDE permite também o fácil uso de *timers*, que são temporizadores que podem ser programados para realizar uma tarefa a cada instante de tempo determinado. Assim, a implementação de rotinas para a interface, como verificação de estados de sensores, comunicação entre cliente e servidor, etc, é

facilitada. Desta forma, para o caso deste projeto, o VB é uma opção bastante vantajosa.

Outra vantagem do VB é a integração direta com o Prodave, cujas bibliotecas (DLLs) foram feitas especialmente para esta linguagem ou para C++.

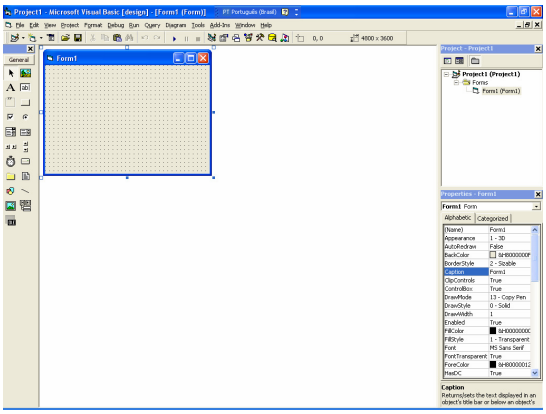

Figura 16 – Janela principal da IDE

2.11.1) Arquivos OCX

Arquivos OCX (“.ocx”) são arquivos de controle especialmente criados para dar suporte ao funcionamento de alguma aplicação do Windows. Podem conter gráficos, textos, imagens, sons, vídeos, ou qualquer outro tipo de informação (WHATIS?.COM, 2006). No caso deste projeto, um arquivo OCX permite a utilização da ferramenta *sockets*.

Para se usar um arquivo OCX, abre-se a guia “Components” no menu “Project” (Figura 17) do IDE. Uma lista de opções é então exibida. Todos os itens desta lista são OCX que já acompanham a IDE para o uso em aplicações mais comuns do Windows. Após marcar a opção selecionada (Figura 18), clica-se em “OK” e o OCX está pronto para ser usado, e surge um novo item na barra de ferramentas (Figura 19).

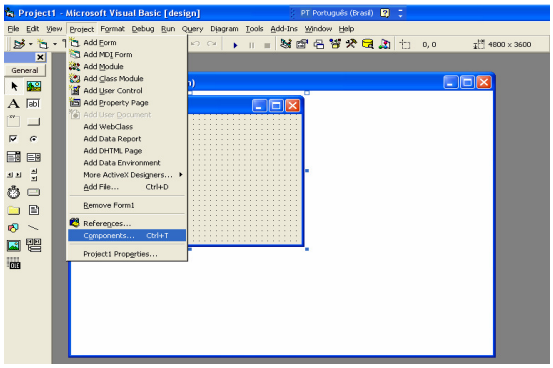

Figura 17 – Guia "Components" no menu "Project"

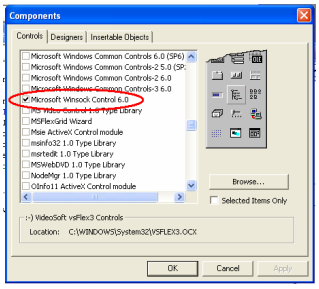

Figura 18 – Lista de OCX disponíveis

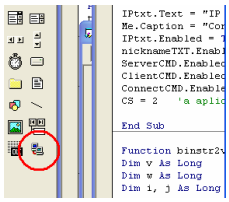

Figura 19 – Após adicionar o OCX uma nova ferramenta é disponibilizada

2.12) Parameterize FM354

Dentre os dispositivos de atuação utilizados em sistemas produtivos tem-se os servomotores que são imprescindíveis para determinadas etapas do processo produtivo. O servomotor é uma máquina síncrona composta por uma parte fixa (o estator) e outra móvel (o rotor). De um servomotor são exigidas certas características dinâmicas, de rotação, torque constante e precisão de posicionamento. Em geral, a qualidade de um servomotor está associada ao torque constante em larga faixa de rotação, uma larga faixa de controle da rotação e alta capacidade de sobrecarga (WIKIPEDIA (b), 2008).

O SP considerado neste trabalho contém dois servomotores do tipo FM354, da empresa Siemens. Para programar os parâmetros destes servomotores no CP, como velocidade, posição, etc, é preciso utilizar um *software* complementar ao STEP7, chamado “Parameterize FM354”.

O Parameterize FM354 permite ao usuário modificar os dados de máquina (tipo de encoder, pontos de referência, mensagens de erro, etc), criar programas para movimentação dos motores (controlando a velocidade e posição por coordenadas absolutas ou incrementais) e também monitorar em tempo real o movimento dos motores, indicando erros e posições atuais (SIEMENS (c), 2008).

Para utilizar o Parameterize FM354, deve-se instalá-lo junto ao STEP7 (neste caso, está se referindo a um programa da Siemens para edição de programas de controle em STEP7). Uma vez instalado, deve-se abrir um projeto no STEP7, selecionar a opção “Hardware” (Figura 20) e incluir o módulo “FM354” (Figura 21).

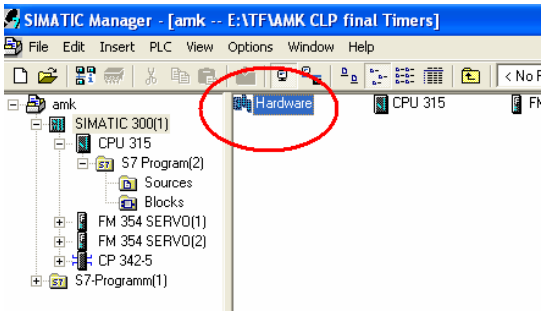

Figura 20 – Configuração do hardware

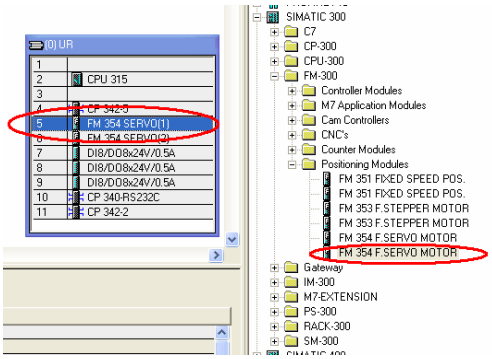

Figura 21 – Inclusão do FM354

Após o módulo ser incluído, clica-se com o botão direito do mouse em “FM354” e seleciona-se “Objetic Properties” (Figura 22). A seguir, clica-se em “Parameter” (Figura 23).

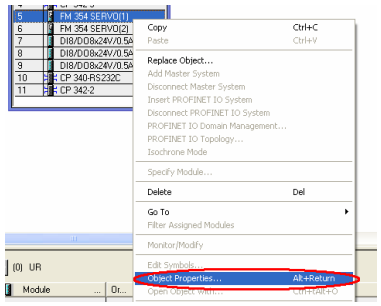

Figura 22 – Object properties

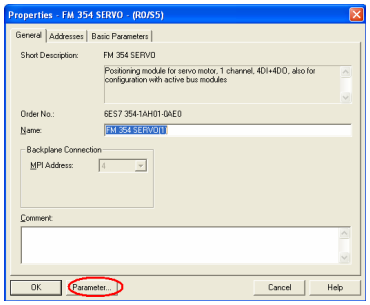

Figura 23 – Propriedades do FM354

Com isso, a janela principal do Parameterize FM354 é aberta (Figura 24). É possível aqui observar diversos botões como “Machine Data”, “StartUp”, “Error Display”, “Traverse Programs”, etc. Para ter acesso às funções destes botões, a CPU do CP deve estar no estado STOP, caso contrário, só é possível fazer a monitoração dos parâmetros do FM354.

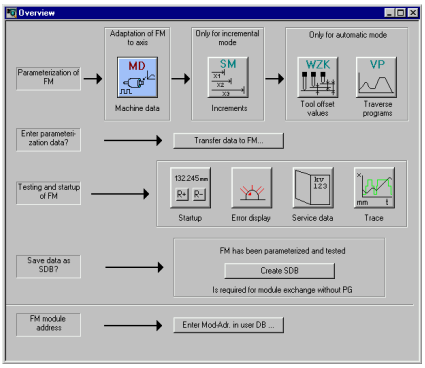

Figura 24 – Janela principal do Parameterize FM354

Ao selecionar “Machine Data”, pode-se modificar os parâmetros do motor para uma aplicação específica. O botão “Increments” só é usado quando programa-se em coordenadas incrementais, e permite a inserção dos incrementos de cada passo. “Tool offset values” é usado para programar compensação de ferramenta. “Traverse programs” permite criar programas em uma linguagem semelhante a linguagem G, e pode-se assim programar em coordenadas absolutas, incrementais, alterar a velocidade, criar *loops*, etc. O botão “Startup” abre uma interface que permite a monitoração da posição e também o controle dos motores. Quando há algum erro durante a movimentação do motor, ao clicar em “Error Display” é exibida uma mensagem informando a causa do erro. “Service Data” serve para detalhar os parâmetros do movimento atual do motor, ou seja, exibe a velocidade, posição, posição do *encoder*, voltagem, erro de medida, etc. Por fim, o botão “Trace” permite a construção de gráficos como “posição x tempo” do motor.

Conforme já foi mencionado, o botão “StartUp” abre uma janela de interface de monitoração e controle do motor (Figura 25). Esta janela possui quatro campos principais: erros; status; dados de entrada; entrada de comandos, configurações e valores para iniciar/parar o motor.

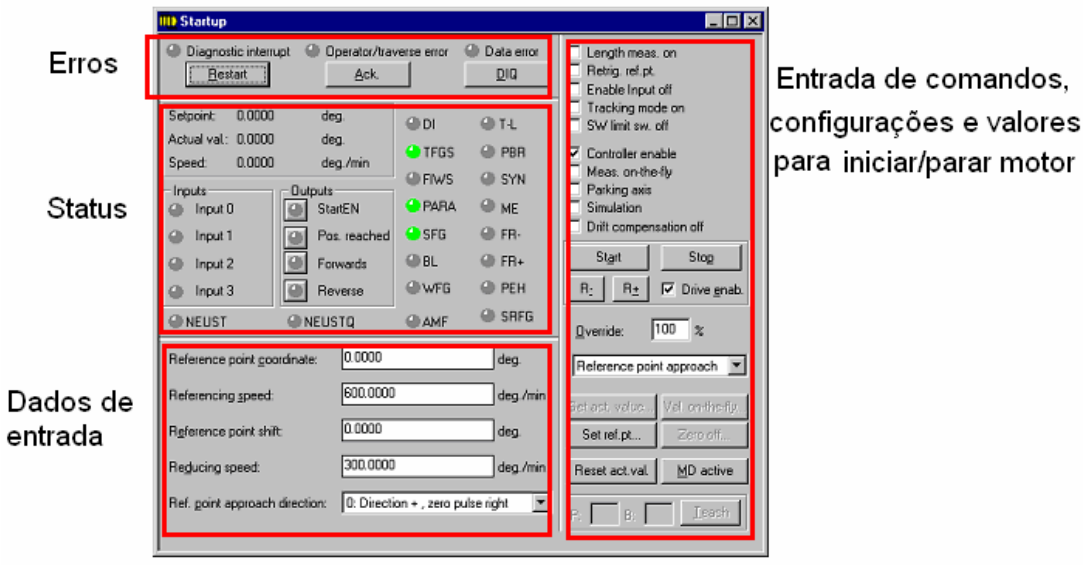

Figura 25 – Janela de StartUp

No campo de erros é informada a causa do erro, e também é onde se reinicia o movimento. Uma vez que um erro é gerado, deve-se clicar em “Restart” ou em “Ack”, possibilitando que a operação do motor continue. O campo de status exibe informações de realimentação como o sentido de rotação do eixo (positiva ou negativa), sincronização com o encoder, posição atual, etc e serve portanto para monitorar os parâmetros. O campo de dados de entrada permite que se insira parâmetros como velocidade, ponto de referência, aceleração, etc. Os dados de entrada variam de acordo com o modo de operação que se seleciona. Esta seleção é feita no campo de entrada de comandos, onde habilita-se os modos de operação (“Jogging”, “Open loop control”, “Reference point approach”, “Incremental relative”, “MDI”, “Automatic” e “Automatic single block”) e também se inicia ou interrompe (Start e Stop) o movimento do motor.

Os modos de operação, selecionados no campo de dados de entrada (Figura 26) são os responsáveis pelos movimentos do motor e deve ser o primeiro parâmetro a ser escolhido, pois de acordo com o modo de operação os demais dados de entrada variam. O modo “Jogging” requer que se insira uma velocidade padrão (que será usada durante todo o movimento do motor) e requer também que a opção “Controller enable”, dentro do campo de dados de entrada, esteja ativada. Deve também selecionar um sentido de rotação do motor, positiva ou negativa. No modo “Open loop control” tensões de diversas magnitudes são especificadas e então usadas para um movimento. Os requisitos dos dados de entrada são semelhantes ao modo “Jogging”.

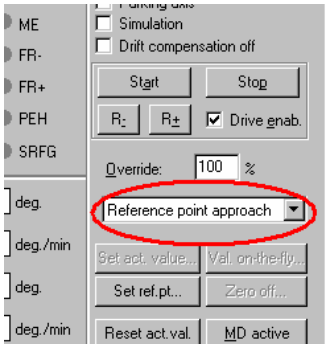

Figura 26 – Seleção dos modos de operação

O modo “Reference point approach” serve para posicionar o motor em um ponto específico determinado em “machine data”, já mostrado anteriormente. Assim, é possível determinar um ponto de referência, sincronizando o eixo, e a partir daí iniciar o movimento do motor. O único requisito para este modo é que “Controller enable” esteja ativado.

Em “Incremental relative” é possível executar posicionamentos em relação a distâncias pré-determinadas. Estas distâncias devem ser programadas em “Increments” (já mencionado anteriormente) onde pode-se ativar o “Controller enable” e selecionar o sentido de rotação (positiva ou negativa) para o incremento.

O modo “MDI” permite que se execute posicionamentos por meio de um programa pré-determinado. Para este modo de operação, “Controller enable” deve estar ativo e o eixo deve estar sincronizado (“Reference point approach”).

Os modos “Automatic” e “Automatic single block”, da mesma forma que o modo “MDI”, utilizam programas pré-determinados. A diferença é a possibilidade de executar o programa completo automaticamente (“Automatic”) ou por blocos (“Automatic single block”), que executa o programa de trechos em trechos (SIEMENS (c), 2008).

É importante destacar que tudo o que se faz manualmente via Parameterize FM354 é possível de ser feito também via programação em STEP7. O software acompanha um *Datasheet* (incluso neste relatório como anexo A) relacionando endereços do módulo FM354 com os parâmetros do Parameterize FM354. Assim, pode-se programar o CP de forma que ative todos os parâmetros necessários para a operação, e também se faça a monitoração destes parâmetros.

Outro ponto importante é que a comunicação entre o software Parameterize FM354 e o módulo FM354 é feita via placas Profibus específicas. No presente projeto, a placa utilizada é a CP5613 A2 (Figura 27), da empresa Siemens. Outros dispositivos como PC Adapters, por exemplo, embora façam a comunicação entre computador e CP, não são compatíveis com o módulo

FM354 e portanto não podem ser usados para a comunicação entre Parameterize FM354 e o módulo FM354 (SIEMENS (d), 2008).

Figura 27 – CP5613 A2 (SIEMENS (e), 2008)

2.13) *Sockets*

Para a comunicação remota entre dois computadores é preciso uma ferramenta que garanta o envio e recebimento de dados entre eles.

Uma das primeiras formas de se desenvolver aplicações distribuídas em um ou mais computadores foi através do uso de *sockets*. *Sockets* são um método para a comunicação entre um programa cliente e um servidor dentro de uma rede de comunicação (AXIS COMMUNICATIONS, 2008). Com isso foram desenvolvidas diversas aplicações cliente/servidor onde cliente(s) e servidor poderiam estar em máquinas diferentes, distantes umas das outras (PINHEIRO, 2006).

O *sockets* se originou como parte do sistema operacional BSD UNIX. O trabalho foi financiado pelo governo americano, através da qual a University of California em Berkeley desenvolveu e distribuiu uma versão de UNIX que continha protocolos de ligação inter-redes TCP/IP. Muitos vendedores de computadores portaram o sistema BSD para seu hardware e o usaram como base em seus sistemas operacionais comerciais (THE JAVA™ TUTORIAL, 2006).

Em se tratando de *hardware*, os *sockets* são como “encaixes” para o processador, isto é, módulos de memória entre outros componentes do

computador. Do ponto de vista de software, os *sockets* são módulos de programas que conectam os aplicativos ao protocolo da rede de comunicação, facilitando o trabalho do programador, que precisa apenas instruir o programa a abrir um dos *sockets* disponíveis. O programa passa então a enviar e receber dados através da rede de comunicação (PINHEIRO, 2006). A Figura 28 mostra uma representação esquemática da utilização de *sockets*:

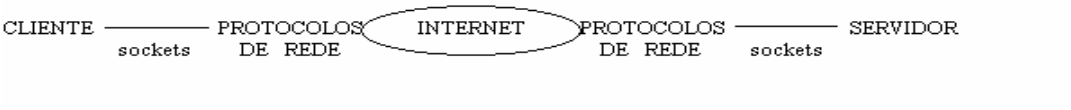

Figura 28 - Esquema representando a comunicação entre servidor-cliente via sockets.

A programação dos *sockets* difere da I/O (Input/Output) convencional porque um aplicativo deve especificar diversos detalhes para usar um *socket*. Por exemplo, um aplicativo deve escolher um protocolo de transporte (UDP, TCP/IP, etc), fornecer o endereço de protocolo (endereço IP, UDP, etc) de uma máquina remota e especificar se o aplicativo é um cliente ou um servidor. Para acomodar todos os detalhes, cada *socket* tem diversos parâmetros e opções - um aplicativo pode fornecer valores para cada um deles (THE JAVA™ TUTORIAL, 2006).

Essencialmente, um aplicativo cria um *socket* e então invoca funções para especificar em detalhes como será usado o *socket*. A vantagem dessa abordagem é que a maioria das funções tem três ou menos argumentos; a desvantagem é que um programador deve se lembrar de chamar múltiplas funções ao usar *sockets* (DONAHOO, 2001).

2.13.1) *Sockets* TCP/IP

O processo de comunicação ocorre da seguinte forma: o servidor escolhe uma determinada porta (*port*) e fica aguardando conexões nesta porta. O cliente deve saber previamente qual a máquina servidora (*host*) e a porta que o servidor está aguardando conexões. O cliente então solicita uma conexão em um host/porta, conforme demonstrado na Figura 29 (NUNES, 2008):

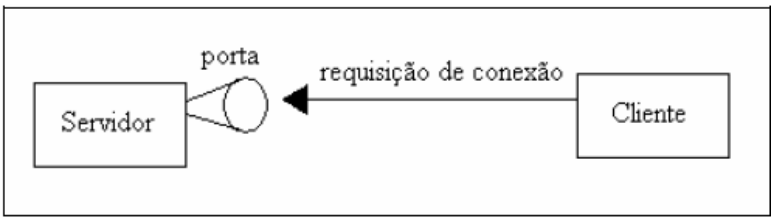

Figura 29 – Início de conexão via sockets.

Se nenhum problema ocorrer, o servidor aceita a conexão, criando assim um canal de comunicação entre cliente e o servidor (Figura 30).

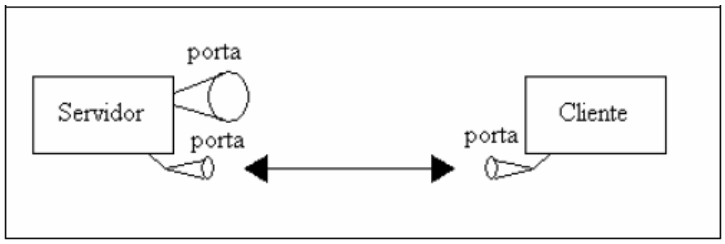

Figura 30 – Canal de comunicação.

Tipicamente o comportamento do servidor é ficar em "*loop*" aguardando novas conexões e gerando *sockets* para atender as solicitações de clientes.

3) Estrutura de Controle do Sistema Produtivo Teleoperado

O projeto aqui consiste em desenvolver as interfaces gráficas para a supervisão de um sistema produtivo (SP) teleoperado. Assim, deve-se programar os controladores programáveis (CPs) de maneira que as estações de trabalho do SP considerado executem o processo / tarefas previstas. Além disso, é necessário estabelecer a comunicação entre os CPs e os computadores, isto é, programar a interface local e remota para que se comuniquem, respectivamente, com outros CPs (via Profibus), com o comutador remoto (*sockets*) e com o computador local (*sockets*).

Para estas tarefas mencionadas, considera-se a estrutura geral da parte de controle e supervisão de um SP, que pode ser dividida em três níveis hierárquicos conforme indicado na Figura 31. No nível diretamente relacionado com os sinais de atuadores e sensores e respectivos dispositivos de controle o foco está na comunicação e programação dos CPs. Um conjunto destes CPs está conectado, em geral, através de algum protocolo de comunicação padrão (como, por exemplo, o Profibus na área industrial), a um supervisor local (computador local) que neste caso atua como um gerenciador específico de uma parte da planta que pode estar instalada eventualmente distante de outro supervisor local.

Através de uma infraestrutura como a internet, considera-se que estes supervisores locais estão conectados com os supervisores remotos (computadores remotos), estabelecendo com isto a devida integração dos gerenciadores de plantas que compõem o SP independente de onde elas de fato estão fisicamente/geograficamente instaladas.

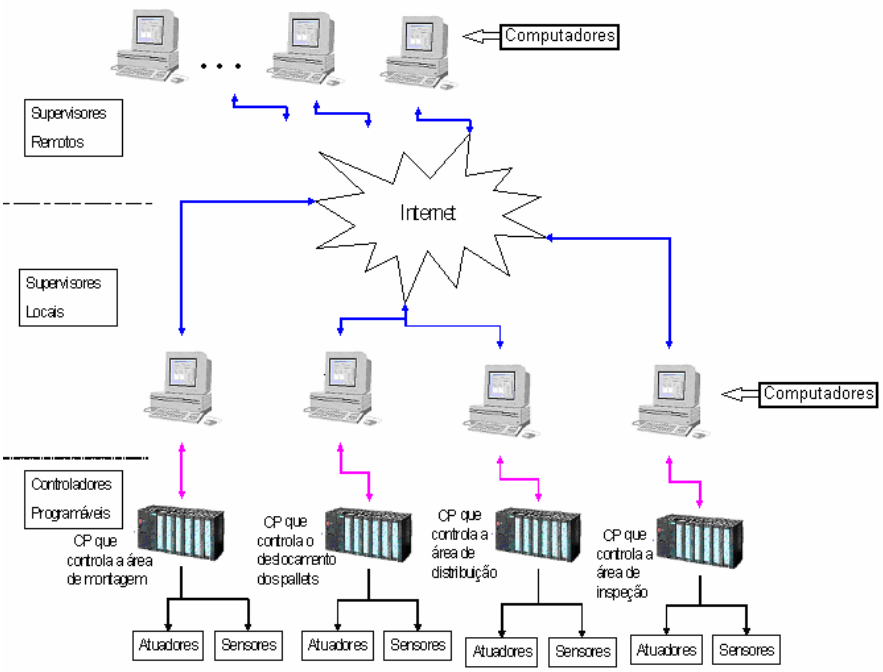

Figura 31 – Estrutura de controle e supervisão de um sistema produtivo.

Considerando o sistema flexível de montagem, citado no capítulo 1, a seguinte simplificação foi adotada para este projeto:

- cada estação de trabalho está emulando um supervisor local, isto é, cada estação de trabalho é vista como uma planta específica com certo grau de autonomia operacional cujas tarefas são gerenciadas por um supervisor local.

Esta simplificação é considerada adequada neste caso, pois o foco do trabalho está nos algoritmos de processos distribuídos através de internet e na interface com os supervisores locais e remotos.

Dessa maneira, seria como se cada estação fosse uma fábrica diferente e dispersa fisicamente, coordenadas por um sistema global, que coordena a produção como um todo. Assim, cada estação é controlada pelo seu supervisor, que por sua vez é controlado pelo supervisor central do sistema.

3.1) Interface implementada

A aplicação computacional desenvolvida para atender os requisitos operacionais apresenta uma arquitetura hierárquica (Figura 32), onde:

- a camada inferior visa a programação do SP, ou seja, o desenvolvimento de um programa de controle em STEP7 que é executada pelo CP. Este programa deve conter toda a lógica das etapas do processo produtivo e portanto esta camada deve-se integrar ASI, Profibus, STEP7 e Parameterize FM354.

- uma camada intermediária permite a comunicação local com o CP. Neste ponto, deve-se integrar o VB com o Prodave, de modo a estabelecer a comunicação via Profibus entre computador local e CP.

- uma camada superior permite a comunicação dispersa geograficamente via internet. *Sockets* devem ser integrados com o VB, de modo que supervisores remotos se comuniquem com suervisores locais.

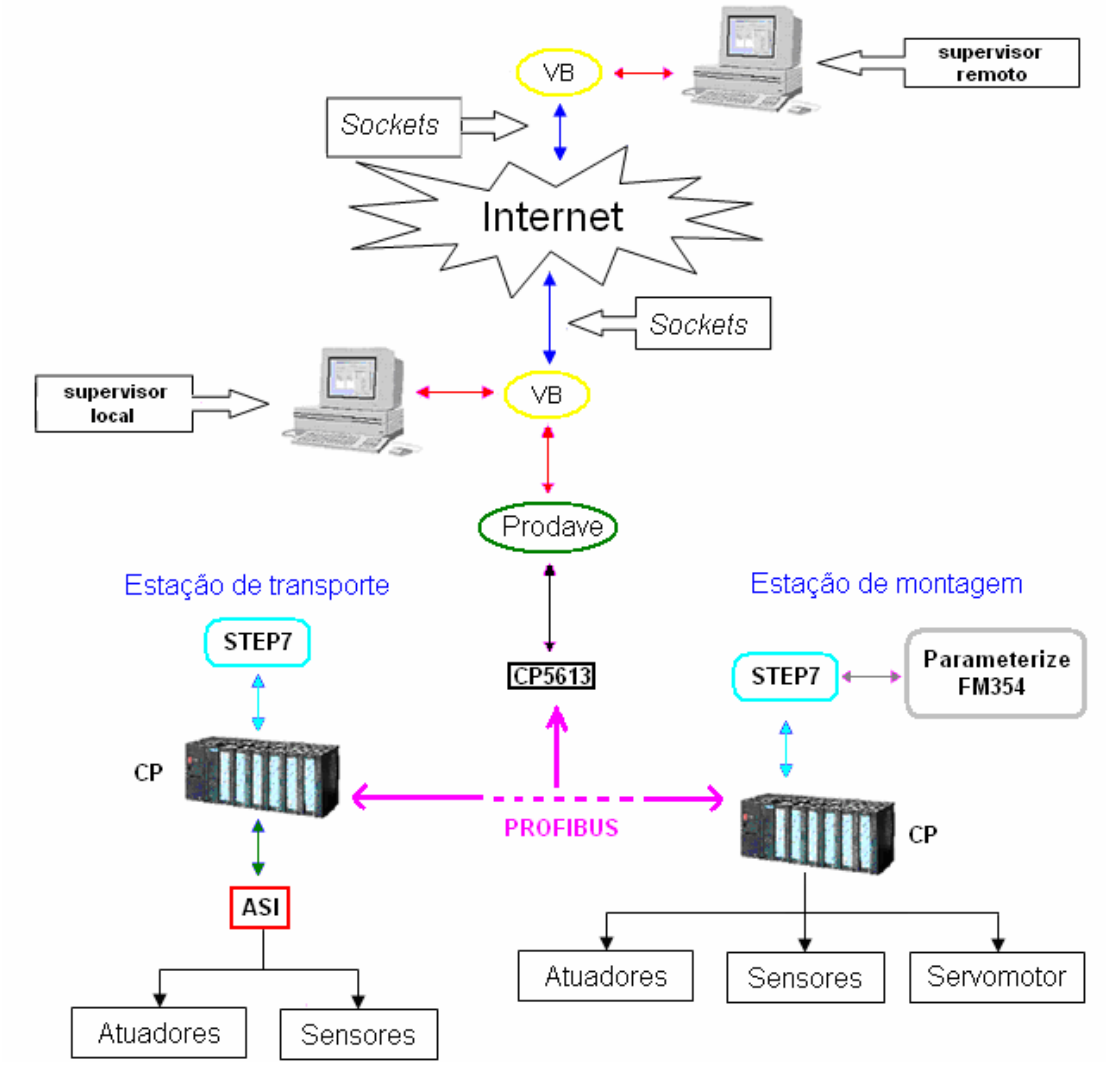

Figura 32 – Estrutura do software a ser implementado.

4) Modos de Operação das Interfaces

Para implementar a teleoperação e a interface gráfica das estações do sistema produtivo (SP) apresentado no Capítulo 1 foi adotada uma estratégia onde inicialmente deve-se ter definições e especificações precisas sobre a estrutura de controle do SP, componentes físicos e lógicos e, processos envolvidos, bem como as formas de comando e parâmetros de monitoração. A seguir, são selecionadas as ferramentas, conceitos e fundamentos adequados para implementar aquilo que é necessário. Assim, torna-se fundamental a questão de como interpretar visualmente todas as possibilidades do SP e do seu sistema de controle. Neste projeto, isto foi realizado através de diagramas de fluxo, envolvendo as principais lógicas dos programas implementados em STEP7 no CP (código incluso neste relatório como Anexo B) e em VB no computador (código incluso neste relatório como Anexo C).

4.1) Seleção do modo de operação

A primeira função a ser considerada no uso das interfaces é a seleção do modo de operação. O fluxograma é representado a seguir (Figura 33):

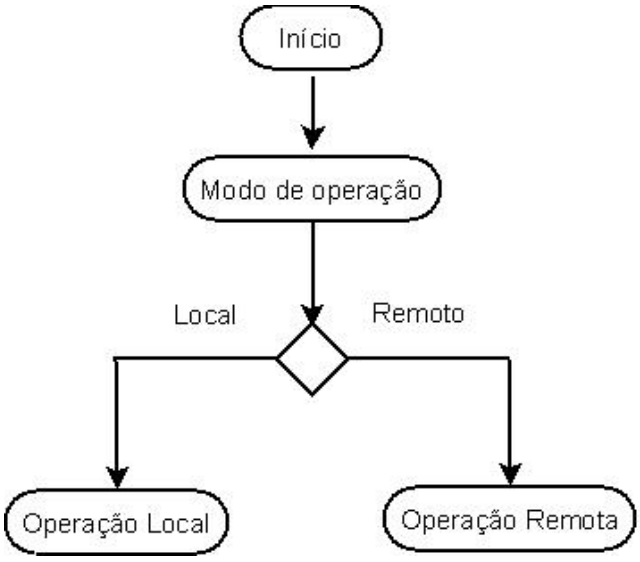

Figura 33 – Escolha da operação.

Nesta etapa, a interface deve apresentar uma opção de operação local ou remota. Na operação local, um operador controla e monitora uma estação do SP por meio de um computador local conectado diretamente ao CP. Na operação remota, a operação é realizada por um operador que está em um local geograficamente diferente do local em que se encontra a estação do SP. Assim, este operador remoto se conecta pela internet ao computador local e teleopera a estação do SP. Nesta operação, é preciso que o computador local tenha uma interface que comunique com a interface do computador remoto, e assim é possível que um segundo operador monitore localmente o sistema via interface local.

4.2) Uso das interfaces

Uma vez que o modo de operação é escolhido, deve-se decidir quais comandos serão disponibilizados nas interfaces. O fluxograma é representado a seguir (Figura 34):

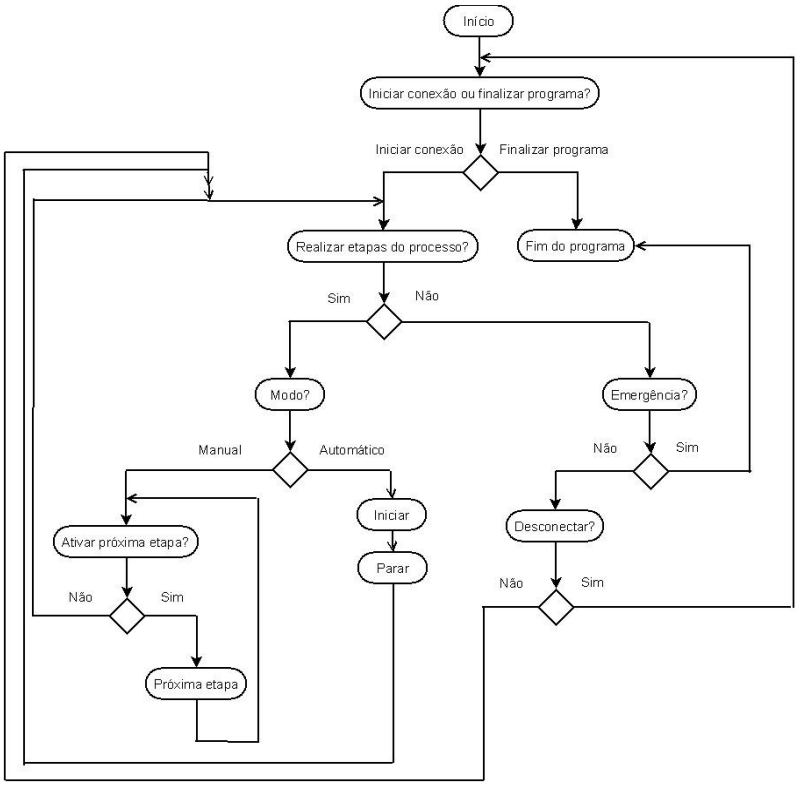

Figura 34 – Uso da interface.

Se o modo de “operação remota” é escolhido, considera-se neste projeto que o operador (local) na interface local pode monitorar o processo e interrompê-lo em caso de emergência (pressionando um botão de emergência na interface). Para o modo de “operação local” os comandos na interface local são os mesmos que na interface remota quando o modo de “operação remoto” é escolhido, já que a diferença está apenas no local geográfico em que o comando está sendo enviado.

Em qualquer um desses casos, o operador pode primeiro decidir entre iniciar a conexão entre computador local e CP ou finalizar o programa da interface. Se iniciar a conexão, a próxima decisão a ser tomada é realizar ou não etapas do processo produtivo. Ao realizar etapas do processo, deve selecionar se a operação será feita automaticamente ou manualmente. Em modo automático, deve decidir quando iniciar e interromper o processo, enquanto que em modo manual deve apenas informar se a próxima etapa deve ser efetuada ou não. Se o operador decide não realizar uma etapa, pode pressionar o botão de emergência, interrompendo o programa. Caso não pressione este botão, pode desconectar o computador local do CP. Se desconectar, volta à situação inicial do programa da interface onde pode então finalizá-lo.

4.3) Etapas da estação de transporte

Nesta seção é detalhada a lógica para as etapas (conjunto de tarefas) da estação de transporte, envolvendo sensores e atuadores. Para entender o fluxograma, considere a nomenclatura adotada na Figura 35. Note que os quatro postos (A, B, C e D), locais no sistema de transporte onde os pallets (carros) podem parar, são similares, ou seja, possuem o mesmo número de sensores e atuadores, e portanto a lógica para um posto pode ser repetida para os demais. Neste caso, ilustra-se o fluxograma apenas para o posto A.

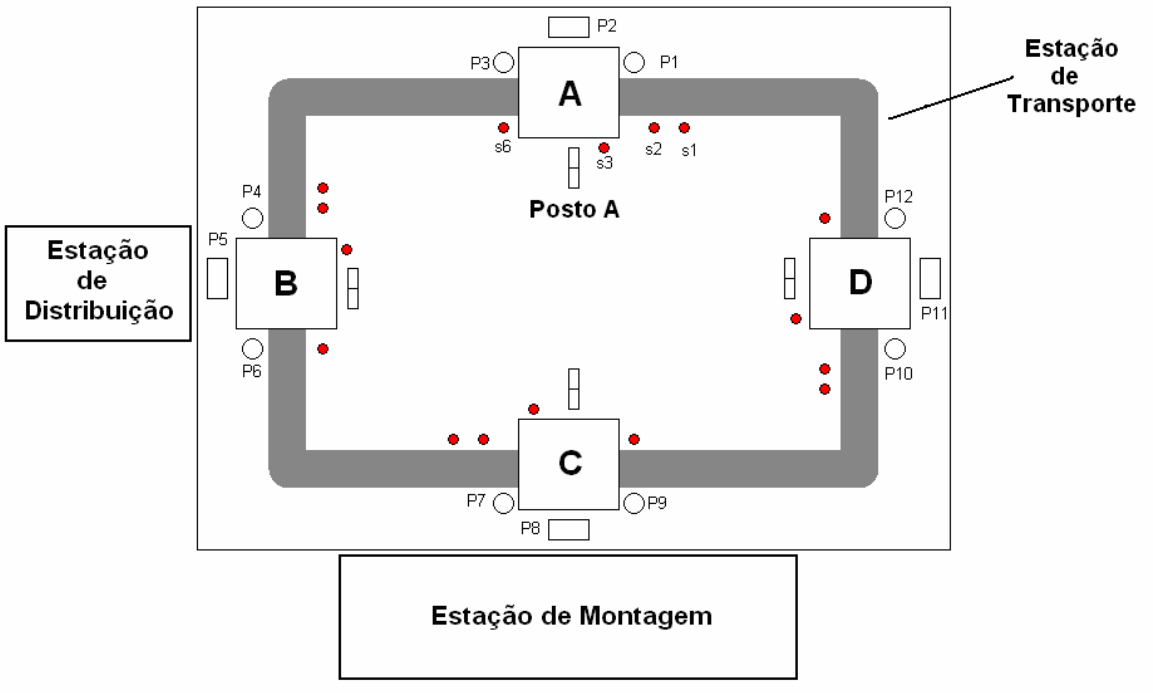

Figura 35 – Nomenclatura adotada para a estação de transporte.

As etapas da estação de transporte podem ser divididas em três principais, sendo a entrada de um carro em um posto, o seu atendimento (recebimento ou liberação de uma peça) e por fim a sua saída. O fluxograma destas etapas é ilustrado na Figura 36.

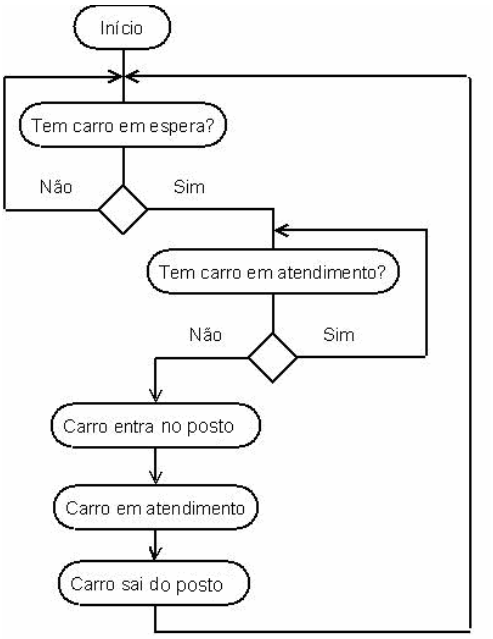

Figura 36 – Etapas da estação de transporte.

Na primeira etapa, o carro fica preso pelo pistão “P1”. Neste momento, o sensor “s1” fica ativo, indicando que há um carro aguardando atendimento. Se não houver outro carro em atendimento (“s3” desativado) o pistão “P1” libera o carro e quando este passa por “s2”, ativa “P1” novamente, colocando o próximo carro em espera. O pistão “P3” fica assim ativado e mantém o carro no posto. Uma vez que o carro está em atendimento, “P2” é ativado e logo que o atendimento termina, o pistão é desativado. “P3” é então desativado e libera a saída do carro, que ao ativar “s6” ativa novamente “P3”. Finalizadas estas três partes, o processo recomeça.

4.4) Etapas da estação de montagem

Nesta seção é detalhada a lógica para as etapas da estação de montagem, envolvendo sensores, atuadores e servomotores.

Para iniciar uma montagem, são necessárias três condições. A primeira é que tenha um carro em atendimento no posto C. Outras condições são que o pistão P8 da estação de transporte esteja ativo e que o carro esteja transportando uma peça. O fluxograma destas etapas é ilustrado na Figura 37.

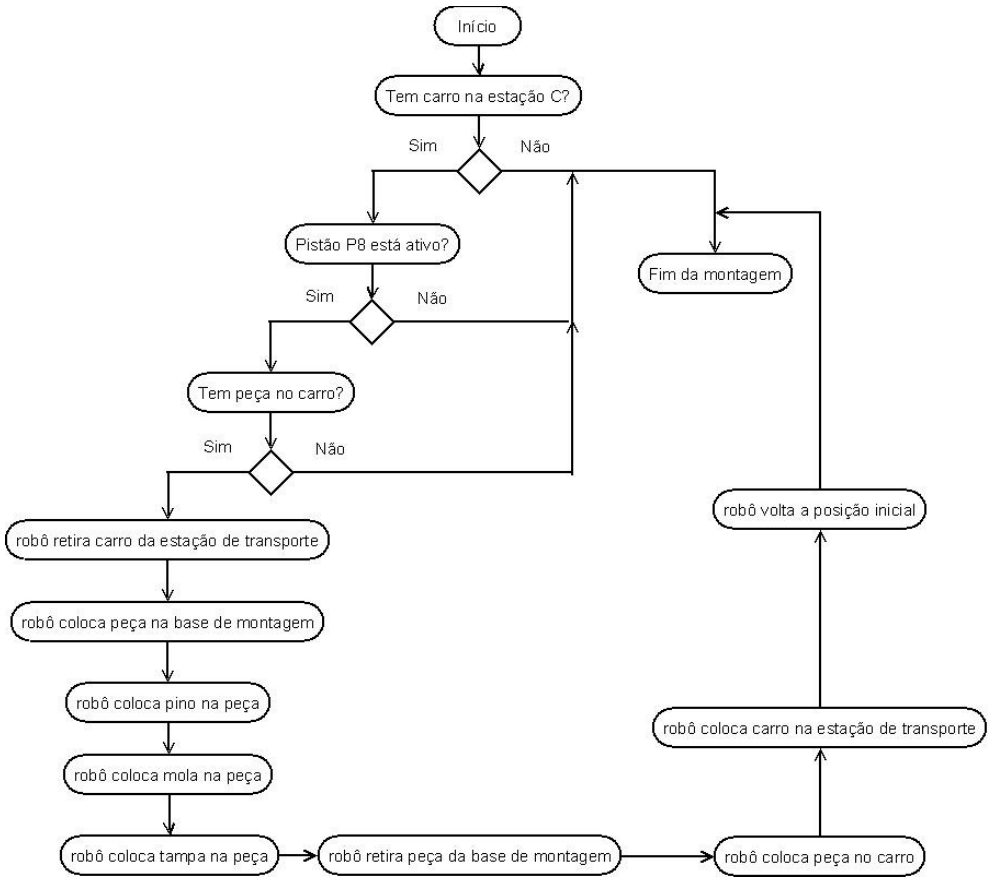

Figura 37 – Etapas da estação de montagem.

A montagem em si é efetuada por um robô manipulador que executa as seguintes tarefas: retira o carro da estação de transporte e o inse na estação de montagem. Feito isto, coloca a peça que estava no carro em uma base de montagem. A seguir insere na peça o pino de cor correspondente, seguido de uma mola e uma tampa. A peça completa é então recolocada no carro e este é devolvido para a estação de transporte.

5) Implementação e Testes

Inicialmente são apresentados uma amostra dos resultados relativos à programação de CPs, seguidos pelos resultados de comunicação entre computador e CP e, pela utilização de *sockets* para comunicar dois computadores através da *Internet*. Na seqüência, são mostrados os resultados da programação da interface (tanto remota como local) e, por fim, os resultados finais do projeto.

5.1) Programação dos CPs

A criação e implementação dos programas dos CPs foi dividida em três partes principais: programação dos sensores e atuadores da estação de transporte, programação dos sensores e atuadores da estação de montagem e programação dos servomotores da estação de montagem.

A idéia é que estas três partes fossem programadas a partir de *flags*, que são ativadas/desativadas via interface em VB. Cada *flag* é responsável por uma etapa do processo produtivo, e desta maneira, a interface em VB envia os sinais correspondentes ativando/desativando cada *flag* em determinada ordem, constituindo o processo produtivo completo.

A princípio, os endereços dos *flags* poderiam ser escolhidos livremente, bem como as suas relações com cada etapa do processo. No entanto, como a interface é responsável por controlar cada etapa, ordenou-se os *flags* e as etapas de forma lógica, facilitando posteriormente a programação da interface. Assim, optou-se por ordenar os *flags* em ordem crescente, cada um representando uma etapa do processo, que é constituído por 67 etapas. Por exemplo, o *flag* de endereço “M1.0” ativa a primeira etapa, o *flag* de endereço “M2.0” ativa a segunda etapa e assim por diante. Adotando esta estratégia, ao programar a interface para controlar cada etapa, basta implementar uma estrutura de *loop*, que ativa os flags com incremento de uma unidade a cada ciclo (M = M + 1) até atingir a etapa 67 (*flag* “M67.0”) e então reiniciar o ciclo.

Além disso, cada etapa deve ter um *flag* para a implementação do botão de emergência na interface. Para isto, foi escolhido o *flag* “M0.0”, que é normalmente fechado e caso seja ativado (apertando o botão de emergência na interface), o contato se abre e a etapa não é executada.

A Figura 38 ilustra uma etapa do processo programada de acordo com a estratégia escolhida. Para alterar o estado do endereço DB2.DBX6.6 (que é descrito a seguir, juntamente com os outros endereços), basta alterar o *flag* “M1.0”. No entanto, se “M0.0” for ativado, esta etapa é interrompida e a saída não varia mesmo alterando “M1.0”.

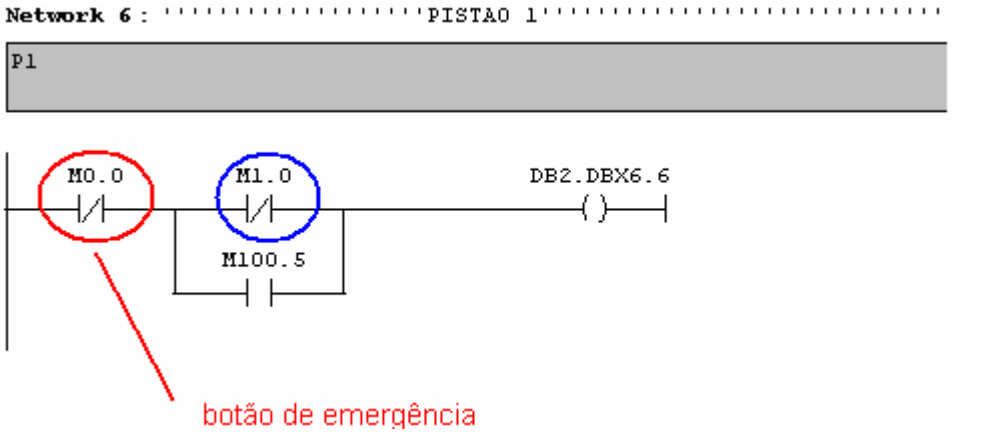

Figura 38 – Exemplo de etapa com a estratégia adotada.

5.1.1) Estação de transporte

Conforme foi descrito no capítulo 2, o ASI é utilizado em ambientes onde existem muitos sensores e atuadores, interligando-os com os CPs de forma relativamente barata e eficiente.

Na estação de transporte, todos os sensores e atuadores estão conectados ao CP via ASI. Os endereços dos sensores estão mostrados na Tabela 5 e os endereços dos atuadores da Tabela 6. A Figura 39 mostra a posição de cada atuador e sensor na estação de transporte.

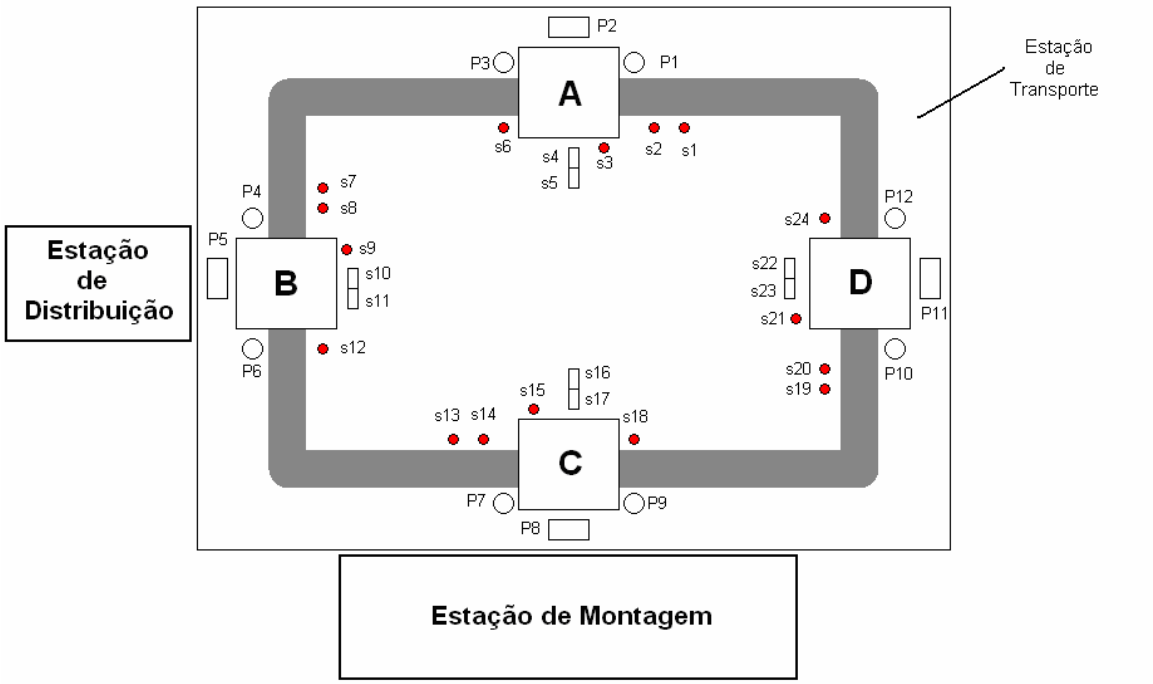

Figura 39 – Nomenclatura adotada para os atuadores e sensores da estação de transporte.

Tabela 5 – Endereços dos sensores da estação de transporte.

DB	DBB	Bit	Descrição (valor)
1	0	0X000000	s12 - (1 ativado, 0 desativado)
1	0	X0000000	s9 - (1 ativado, 0 desativado)
1	0	00X00000	s11 - (1 ativado, 0 desativado)
1	0	000X0000	s10 - (1 ativado, 0 desativado)
1	1	00000X00	s8 - (1 ativado, 0 desativado)
1	1	0000X000	s7 - (1 ativado, 0 desativado)
1	2	X0000000	s13 - (1 ativado, 0 desativado)
1	2	0X000000	s14 - (1 ativado, 0 desativado)
1	2	0000X000	s15 - (1 ativado, 0 desativado)
1	2	00000X00	s18 - (1 ativado, 0 desativado)
1	2	000000X0	s17 - (1 ativado, 0 desativado)
1	2	0000000X	s16 - (1 ativado, 0 desativado)
1	3	X0000000	s21 - (1 ativado, 0 desativado)
1	3	0X000000	s24 - (1 ativado, 0 desativado)
1	3	00X00000	s22 - (1 ativado, 0 desativado)
1	3	000X0000	s23 - (1 ativado, 0 desativado)
1	4	0000X000	s19 - (1 ativado, 0 desativado)
1	4	00000X00	s20 - (1 ativado, 0 desativado)
1	5	X0000000	s1 - (1 ativado, 0 desativado)
1	5	0X000000	s2 - (1 ativado, 0 desativado)

1	5	0000X000	s3 - (1 ativado, 0 desativado)
1	5	00000X00	s6 - (1 ativado, 0 desativado)
1	5	000000X0	s4 - (1 ativado, 0 desativado)
1	5	0000000X	s5 - (1 ativado, 0 desativado)

Tabela 6 – Endereços dos atuadores da estação de transporte.

DB	DBB	Bit	Descrição (valor)
2	1	X0000000	P6 - (0 sobe, 1 desce)
2	1	0X000000	P5 - (1 sobe, 0 desce)
2	1	00X00000	P4 - (0 sobe, 1 desce)
2	3	0000X000	P9 - (0 sobe, 1 desce)
2	3	00000X00	P8 - (1 sobe, 0 desce)
2	3	000000X0	P7 - (0 sobe, 1 desce)
2	4	X0000000	P12 - (0 sobe, 1 desce)
2	4	0X000000	P11 - (1 sobe, 0 desce)
2	4	00X00000	P10 - (0 sobe, 1 desce)
2	6	0000X000	P3 - (0 sobe, 1 desce)
2	6	00000X00	P2 - (1 sobe, 0 desce)
2	6	000000X0	P1 - (0 sobe, 1 desce)

A partir desses endereços, implementou-se a lógica mostrada no capítulo 4 apenas para um dos postos (A, B, C ou D) de atendimento dos carros, sendo que os demais postos são programados analogamente.

Os *flags* responsáveis pelos quatro postos da estação de transporte são “M5.0”, “M6.0”, “M7.0” e “M8.0”. Para testar o programa implementado, cada um desses *flags* foi ativado manualmente via STEP7, utilizando o botão “monitor” ilustrado na Figura 40.

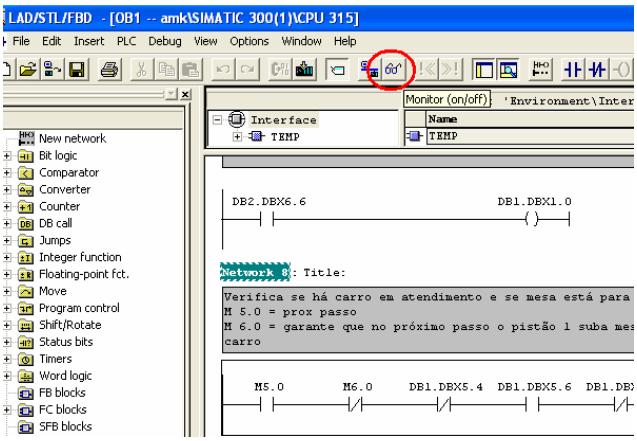

Figura 40 – Botão “monitor” do STEP7.

Uma vez que este botão é ativado, pode-se comunicar diretamente com o programa no CP, similarmente ao Prodave. Para alterar uma variável, basta clicar com o botão direito e escolher entre modificar o valor para “1” ou para “0” (Figura 41).

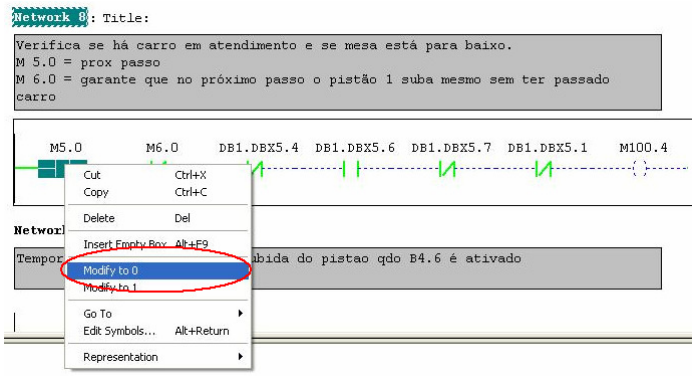

Figura 41 – Modificando o valor da variável.

Ao alterar o valor, é possível observar o estado da saída. No exemplo da Figura 41, supondo que o valor seja alterado para “1”, é possível observar que uma linha verde é estabelecida entre os contatos (que estão fechados) e a saída “M100.4” é ativada (Figura 42).

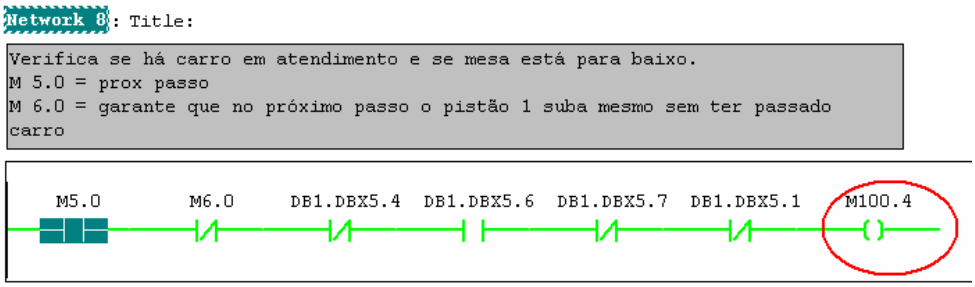

Figura 42 – Saída ativada.

5.1.2) Estação de montagem (sensores e atuadores)

Na estação de montagem, os sensores e atuadores estão endereçados diretamente na memória do CP e portanto não utilizam ASI como a estação de transporte. Desta forma, utilizando os endereços da Tabela 7, foram testadas individualmente cada uma das etapas do processo produtivo. Note que no STEP7 a letra “E” representa endereços de entrada (sensores) e a letra “A”

endereços de saída (atuadores). Estas nomenclaturas podem variar de acordo com o idioma (inglês ou alemão) que está sendo usado no STEP7. As letras “E” e “A” são usadas para o idioma alemão, e para o inglês, a letra “E” se torna “I” e “A” se torna “Q”.

Tabela 7 – Endereços dos atuadores e sensores da estação de montagem.

Byte (A - atuador, E - sensor)	Bit	Descrição (valor)
A20	X0000000	mostra pino preto (1)
A20	0X000000	mostra pino prata (1)
A20	00X00000	pistão mola (1 dentro, 0 fora)
A20	0000X000	pistão tampa dentro (1)
A20	00000X00	pistão tampa fora (1)
A20	0000000X	peça (1 solta, 0 presa)
A16	X0000000	fecha garra (1)
A16	0X000000	abre garra (1)
A16	00X00000	garra a 0^o (1)
A16	000X0000	garra a 270^o (1)
A16	0000X000	garra move Z+ (1)
A16	00000X00	garra move Z- (1)
E16	X0000000	garra fechada (1)
E16	0X000000	garra aberta (1)
E16	00X00000	garra a 0^o (1)
E16	000X0000	garra a 270^o (1)
E16	0000X000	altura garra nível B2.4 (1)
E16	00000X00	altura garra nível B2.5 (1)
E16	000000X0	altura garra nível B2.6 (1)
E16	0000000X	altura garra nível B2.7 (1)
E20	000X0000	mola fornecida (1)
E20	00000X00	pistão tampa frente (1)
E20	000000X0	tampa fornecida (1)

Como o número de atuadores é relativamente grande, é mais simples realizar o teste pelo STEP7 ativando manualmente cada *flag* por meio de uma tabela de variáveis. A tabela tem funções semelhantes ao botão “monitor”

mostrado anteriormente, mas é mais adequado quando muitos endereços estão envolvidos.

Para criar uma tabela de variáveis, deve-se clicar com o botão direito em “CPU315”, “Insert New Object” e “Variable Table”, conforme a Figura 43.

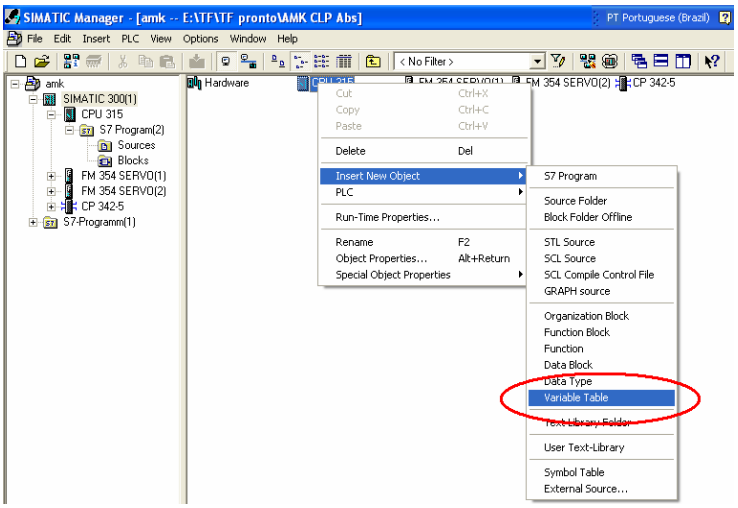

Figura 43 – Tabela de variáveis.

Os endereços que se deseja monitorar/controlar são inseridos nesta tabela (no caso, os endereços descritos na Tabela 7). Pode-se ainda escolher o tipo de dados (booleano, inteiro, word, etc) e também o valor do endereço. Para interagir diretamente com o programa no CP, clica-se no botão “monitor” (Figura 44), e uma vez conectado ao CP, é possível monitorar (Status value) e controlar (Modify value) todos os endereços inclusos na tabela.

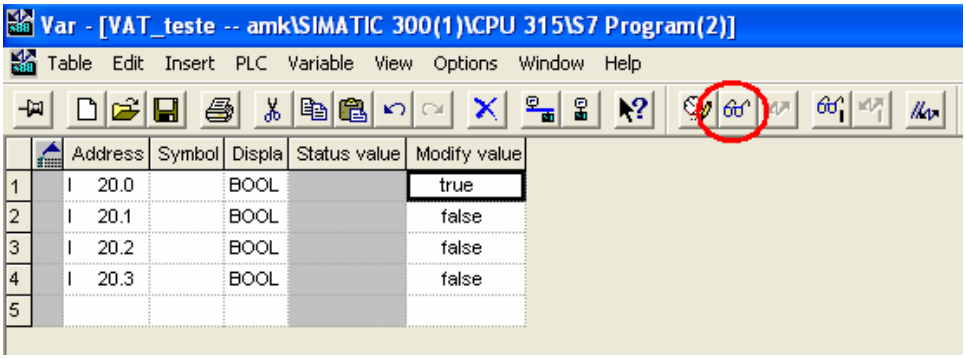

Figura 44 – Controle e monitoração de endereços do CP.

Assim, foram testados os programas para ativar e desativar os atuadores da estação de montagem, bem como ler os estados dos sensores.

5.1.3) Estação de montagem (servomotores)

Para o teste e programação dos servomotores, foi utilizado o software "Parameterize FM354".

O primeiro passo foi determinar as coordenadas das posições referenciadas e a serem mantidas pelos motores (1 e 2, conforme mostra a Figura 45) que movimento o robô manipulador ao longo do processo produtivo. A Figura 45 ilustra as posições da garra (compostas pela combinação entre as coordenadas dos dois motores) na estação de montagem e a Tabela 8 descreve as coordenadas de cada posição.

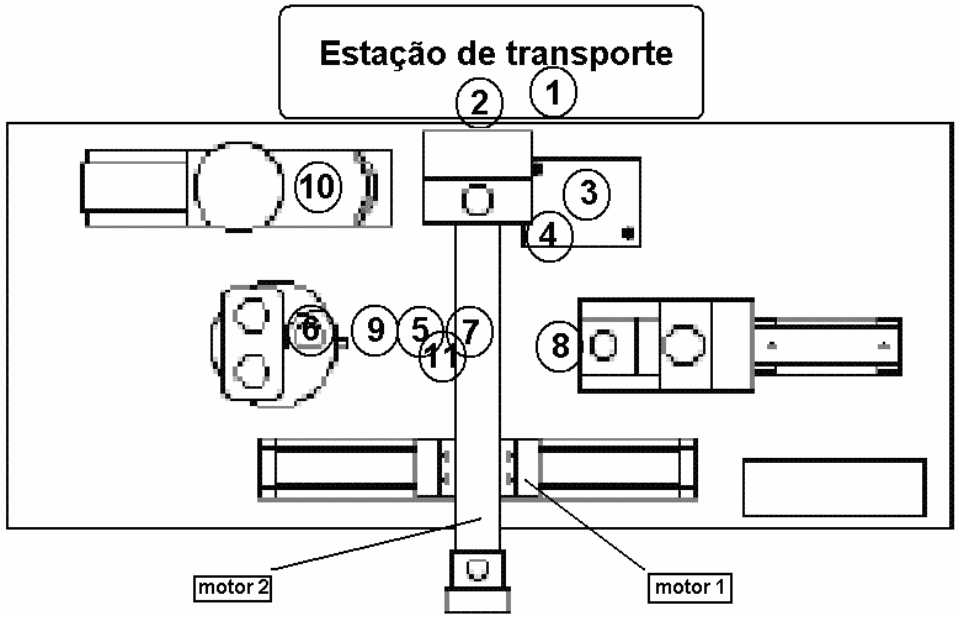

Figura 45 – Possíveis posições da garra do robô manipulador.

Tabela 8 – Coordenadas dos motores do robô manipulador.

Posição	motor 1(mm)	motor 2 (mm)	Descrição
1	70	80	Configuração inicial
2	238.395	55.49	Garra sobre estação de transporte
3	106.64	268.16	Coloca carro na estação de montagem
4	213.95	308.9	Retira peça do carro

5	252.59	407.425	Coloca peça na base de montagem
6	485.25	437.355	Pega pino
7	177.92	409.385	Coloca pino na base de montagem
8	13.145	429.385	Pega mola
9	321.18	406.14	Coloca mola na base de montagem
10	502.635	243.815	Pega tampa
11	251.32	408.88	Coloca tampa na base de montagem
12 = 4	213.95	308.9	Coloca peça no carro
13 = 3	106.64	268.16	Retira carro da estação de montagem
14 = 2	238.395	55.49	Garra sobre estação de transporte

Após definidas as coordenadas, foi desenvolvido um programa em Parameterize FM354 para cada motor, conforme listado abaixo:

- Programa motor 1:

```
%1 amk
N1 G90 G30 X106.640 F4000.000 M0
N2 G90 X213.950 M0
N3 G90 X252.590 M0
N4 G90 X485.250 M0
N5 G90 X177.920 M0
N6 G90 X13.145 M0
N7 G90 X321.180 M0
N8 G90 X502.635 M0
N9 G90 X251.320 M0
N10 G90 X213.950 M0
N11 G90 X106.640 M0
N12 G90 X238.395 M0
N13 M18
```

- Programa motor 2:

```
%1 amk
N1 G90 X268.160 F3000.000 M0
N2 G90 X308.900 M0
```

N3 G90 X407.425 M0
N4 G90 X437.355 M0
N5 G90 X409.505 M0
N6 G90 X429.385 M0
N7 G90 X406.140 M0
N8 G90 X243.815 M0
N9 G90 X408.880 M0
N10 G90 X308.900 M0
N11 G90 X268.160 M0
N12 G90 X55.490 M0
N13 M18

Os dois programas receberam o nome “amk”, mas os nomes dos programas podem ser definidos livremente pelo programador. O número “1” antes deste nome serve para identificação, ou seja, supondo que diversos programas tenham sido feitos, é possível escolher entre um programa e outro fazendo a chamada por este número escolhido. A letra “N” define cada linha de programação. “G90” representa que as coordenadas em “X” são absolutas e “F” define a velocidade dos motores. “G30” significa que o parâmetro *override* vale 100% e “M0” representa o fim de uma linha de programação. Neste projeto, os motores devem repetir as tarefas inúmeras vezes (da mesma forma que um sistema produtivo real) e portanto os programas devem ficar em *loop*. Para isto, foi utilizado o comando “M18” nas últimas linhas dos programas.

Para testar o programa, foi utilizada a interface do software Parameterize FM354. O primeiro requisito é sincronizar o eixo do motor (utilizando *Reference point approach*), pois o programa está em coordenadas absolutas e o motor deve ser informado qual o ponto de origem do movimento. Uma vez que o eixo está sincronizado, deve-se ativar “*Controller enable*”, “*Drive Enable*”, escolher o modo “*Automatic block sequence*” em seguida escolher o número do programa, que neste exemplo, é “1”.

Para ativar cada etapa do motor, basta clicar o botão “START” (Figura 46). Este botão só pode ser clicado se o motor terminar o movimento anterior, e caso contrário, surge um erro informando que foi enviado um comando para movimentar um motor que já está em movimento.

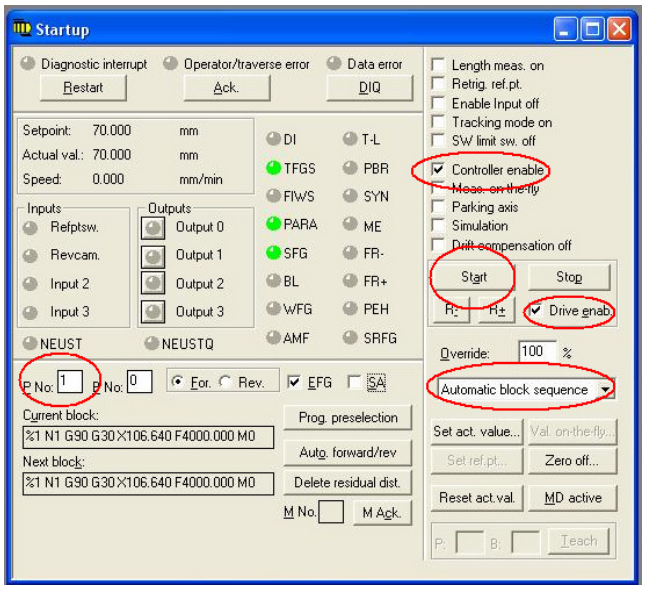

Figura 46 – Teste dos motores.

Após fazer o teste com os dois motores, confirmou-se que a garra do robô atinge os pontos descritos na Tabela 8.

Para programar em *ladder* os passos executados manualmente com a interface do Parameterize FM354, utilizou-se o *datasheet* do FM354 no Anexo A.

Um ponto importante na programação é o uso do botão “Start”, que de acordo com o *datasheet* do FM354, tem como endereço correspondente “DBX15.0”. Para ativar o motor via *ladder*, a seguinte lógica foi adotada (Figura 47):

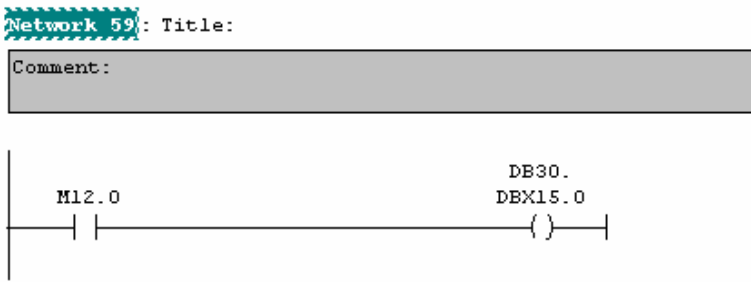

Figura 47 – Lógica em ladder para ativar motores.

A idéia é que o “START” seja ativado quando “M12.0” é ativado, movimentando então o motor. No entanto, esta lógica não funciona, pois ao modificar o valor de “M12.0” para “1”, seria como clicar diversas vezes no botão “START”, e como já foi mencionado anteriormente, isto gera um erro pois estaria se ativando um motor que já está ativado. A solução seria ativar “M12.0” e logo em seguida desativá-lo. No entanto, neste intervalo de tempo (dt), mesmo sendo muito pequeno, pode ser interpretado como vários comandos de “START” (Figura 48), e o problema permanece.

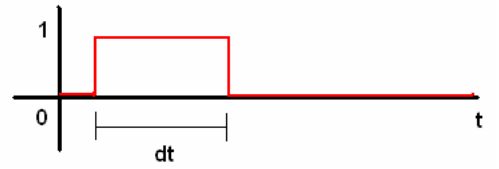

Figura 48 – Infinitos comandos no intervalo de tempo “dt”.

Assim, para garantir que um único comando seja enviado de cada vez, utiliza-se o comando “pulse”, que envia um único pulso para a saída (Figura 49).

Figura 49 – Comando “pulse”

A programação correta é então a ilustrada na Figura 50:

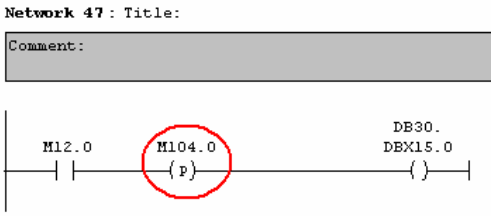

Figura 50 – Programação correta para START.

Agora, ao ativar “M12.0” “M104.0” é ativado e o comando “START” é enviado uma única vez, fazendo com que os motores se movimentem corretamente.

Uma vez desenvolvidos os programas de testes para as estações de transporte e de montagem, estes foram integrados para realizar o teste do processo produtivo completo. Para isto, foi criada uma tabela de variáveis com todas as etapas do processo, ativando-as consecutivamente, até concluir o ciclo.

As etapas e os *flags* correspondentes estão listados na Tabela 9 abaixo:

Tabela 9 – Etapas do processo produtivo.

Etapa	Flag	Descrição
0	M0.0	Botão de emergência
1	M1.0	Configuração inicial estação de transporte
1	M1.1	Sobe garra até posição inicial
1	M1.2 a M1.7	Configuração inicial servomotores
2	M2.0	Configuração inicial carros
3	M3.0	Mostra pino preto (comando inserido a quaquer momento pelo usuário)
3	M3.1	Mostra pino prata (comando inserido a quaquer momento pelo usuário)
4	M4.0	Acknowledge all (desabilita mensagens de erros)
5	M5.0	Carro entra em atendimento
6	M6.0	Pistão eleva carro
7	M7.0	Pistão abaixa carro
8	M8.0	Carro sai de atendimento
9	M9.0	Desce garra
10	M10.0	Fecha garra
11	M11.0	Sobe garra
12	M12.0	Move até posição 3
13	M13.0	Desce garra
14	M14.0	Abre garra
15	M15.0	Sobe garra
16	M16.0	Move até posição 4
17	M17.0	Desce garra
18	M18.0	Fecha garra
19	M19.0	Sobe garra
20	M20.0	Move até posição 5
21	M21.0	Desce garra
22	M22.0	Abre garra
23	M23.0	Prende peça na base de montagem
24	M24.0	Sobre garra
25	M25.0	Gira 270^{o}
26	M26.0	Move até posição 6
27	M27.0	Desce garra

28	M28.0	Fecha garra
29	M29.0	Sobe garra
30	M30.0	Move até posição 7
31	M31.0	Desce garra
32	M32.0	Abre garra
33	M33.0	Sobe garra
34	M34.0	Move até posição 8
35	M35.0	Fornece mola
36	M36.0	Desce garra
37	M37.0	Fecha garra
38	M38.0	Sobe garra
39	M39.0	Move até posição 9
40	M40.0	Desce garra
41	M41.0	Abre garra
42	M42.0	Sobe garra
43	M43.0	Gira 0^{o}
44	M44.0	Move até posição 10
45	M45.0	Fornece tampa
46	M46.0	Retorna pistão tampa
47	M47.0	Desce garra
48	M48.0	Fecha garra
49	M49.0	Sobe garra
50	M50.0	Move até posição 11
51	M51.0	Gira 270^{o}
52	M52.0	Desce garra
53	M53.0	Gira 0^{o}
54	M54.0	Solta peça da base de montagem
55	M55.0	Sobe garra
56	M56.0	Move até posição 12 = 4
57	M57.0	Desce garra
58	M58.0	Abre garra
59	M59.0	Sobe garra
60	M60.0	Move até posição 13 = 3
61	M61.0	Desce garra
62	M62.0	Fecha garra
63	M63.0	Sobe garra
64	M64.0	Move até posição 14 = 2
65	M65.0	Desce garra
66	M66.0	Abre garra
67	M67.0	Sobe garra

5.2) Teste do Prodave

O teste da integração do Prodave com o programa da interface consiste em duas partes principais: a escrita e a leitura de dados no CP.

5.2.1) Função de escrita de dados no CP

Para a escrita, foram utilizados os seguintes códigos:

```
AMOUNT = 1
str = 10000000
value_byte(0) = binstr2value(str)
res = m_field_write(no, AMOUNT, value_byte(0))
```

A variável “AMOUNT” representa a quantidade de valores a serem escritos de uma vez no CP. Neste trabalho, este valor foi sempre “‘1”. A variável “str” indica o valor do *byte* a ser escrito. O valor “10000000” significa que o bit número 0 (em Prodave o primeiro bit à esquerda é o menos significativo) será alterado para “1” enquanto que os demais sete *bits* são alterados para “0”. Na segunda linha do código, a função “binstr2value” transforma a *string* “str” em *bits* e armazena na variável “value_byte”. A variável “no” representa o número do *byte* a ser escrito. A última linha representa a função de escrita, ou seja, é esta função que de fato escreve os valores na memória do CP. Ao fazer uma escrita, o CP retorna um valor ao PC e a variável “res” armazena este valor. Se o valor de “res” for “0”, significa que a escrita foi realizada com sucesso, caso contrário, um erro foi gerado.

A função acima foi testada com um programa ilustrado na Figura 51. Neste teste, a idéia é ativar “M102.0” que em seguida ativa “Q16.0”, que é o endereço para fechar a garra do robô manipulador.

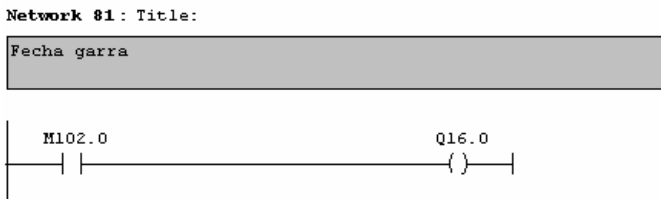

Figura 51 – Programa teste para a escrita de dados no CP.

Utilizando novamente o botão “monitor” descrito anteriormente, foi possível observar que “M102.0” foi ativado com sucesso e a garra foi fechada. Observou-se também que a variável “res” recebeu o valor “0”, confirmando que a escrita foi efetivada.

5.2.2) Função de leitura de dados do CP

Para a leitura, foram utilizados os seguintes códigos:

```
no = 16
res = e_field_read(no, AMOUNT, value_byte(0))
For i = 0 To (AMOUNT - 1)
   s = ""
    j = 1
      s1 = ""
      While (j < 256)
            k = j And value_byte(i)
             If (k <> 0) Then
                s1 = s1 + "1"
                Else
                s1 = s1 + "0"
             End If 'if do k
             j = j * 2
          Wend
   s = s + s1
```

Next i

A leitura tem início com a função “e_field_read”. Caso a leitura tenha sido feita corretamente (res = 0), a função lê o *byte* “no” e armazena o valor em “value_byte”. Em seguida, os *bits* são lidos um por vez e são transformados em uma *string* “s1”. Após todos os *bits* serem lidos, a *string* “s” é retornada com os valores de todos os *bits* do *byte* correspondente.

Para o teste de leitura, foi adotado o mesmo exemplo que o teste de escrita (Figura 51). Uma vez que “M102.0” é ativado “Q16.0” é ativado e o estado deste endereço deve mudar para “1”. Utilizando o código implementado, para ler o valor do *byte* 16 devemos usar “no = 16” e então observar o valor do *bit* “0” deste *byte*. O valor da string “s” obtido foi “10000000”. Como o valor do *bit* “0” foi “1”, pode-se concluir que o teste ocorreu como o esperado.

5.3) Teste da interface no computador remoto

A primeira etapa do teste de operação da interface no computador remoto foi concebida com a implementação de um programa em *sockets* que permite a comunicação entre dois computadores via internet. Os computadores trocam dados durante a operação do sistema produtivo, e os operadores das interfaces também devem estar aptos a se comunicar durante o processo. Assim, o programa deve ter a estrutura de um *chat* (Figura 52), que permite uma comunicação eficaz entre computadores.

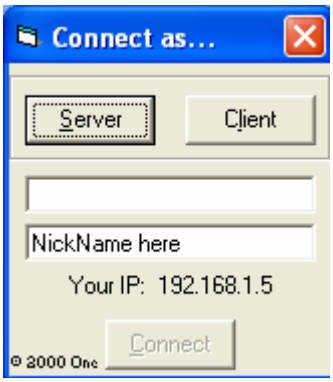

Figura 52 – Chat implementado para teste.

5.3.1) Teste do *chat*

De acordo com o conceito de *sockets* ilustrado no capítulo 2, para iniciar a comunicação é preciso que o servidor inicie a escuta em uma determinada porta. Para isto, após clicar no botão “Server”, deve-se clicar em “Connect”. O cliente, após clicar em “Client”, deve inserir o IP do servidor e em seguida clicar em “Connect”, acessando então a porta em que o servidor está aguardando uma comunicação. Se a conexão for estabelecida corretamente, a janela de *chat* é aberta e a troca de dados pode ser iniciada.

Para enviar uma mensagem utilizando *sockets*, foi utilizado o seguinte comando:

Winsock.SendData dataTXT.Text

Com este comando, a mensagem (*string*) contida na variável “dataTXT” é enviada, e para o receptor receber a mensagem foi implementado o seguinte comando:

Call Winsock.GetData(strData, vbString)

Este comando recebe a mensagem e armazena na variável “strData”.

Com estes códigos foi possível estabelecer a troca de dados entre computadores local (Figura 53) e remoto (Figura 54), e testar o uso de *sockets*.

Figura 53 – Janela de chat do servidor (PC local).

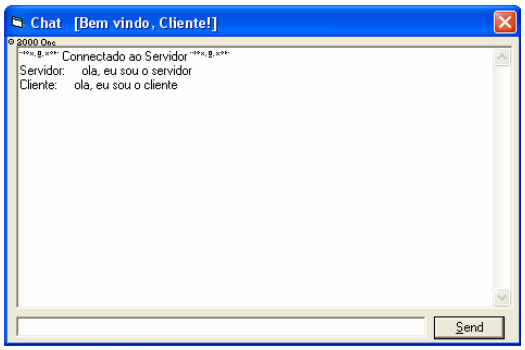

Figura 54 – Janela de chat do cliente (PC remoto).

5.3.2) Teste de teleoperação e monitoração

Do ponto de vista da teleoperação e monitoração, as interfaces devem trocar informações constantemente sobre os estados de sensores e atuadores. Essa troca de informações deve ocorrer em um intervalo relativamente curto de tempo e além disto deve-se garantir que informações não sejam perdidas ou repetidas, pois caso contrário pode prejudicar a tomada de decisões pelo operador e causar erros no processo produtivo.

Quando as interfaces trocam dados, elas enviam e recebem informações da mesma forma que os operadores fazem quando se comunicam via *chat*. Seguindo este princípio, esta estrutura também foi implementada para as operações de monitoração e teleoperação.

Por exemplo, se o operador remoto deseja iniciar o processo, clica no botão “Carregar”. Ao clicá-lo a interface remota envia a mensagem “load” para a interface local, e esta, ao receber uma mensagem, realiza determinada tarefa. Para teste, os códigos implementados foram os seguintes:

```
If strData = "load" Then
    plcadr(0).adr = 2
    plcadr(0).SEGMENTID = 0
    plcadr(0).RACKNO = 0
    plcadr(0).SLOTNO = 2
    plcadr(1).adr = 0
```

```
  VerbIdx = 0
  res = load_tool(1, "S7ONLINE", plcadr(0))
  If (res = 0) Then
  msg.Text = "ok"
  End If 'res = 0
  If (res <> 0) Then msg.Text = "cerror" 'erro de conexão
 End If 'if = "load"
```

A *string* “strData” é a mensagem recebida pela interface local. Se a *string* recebida for “load” executa-se a conexão entre computador local e CP, retornando uma mensagem à interface remota de acordo com o resultado da operação. Se a conexão entre computador local e CP for estabelecida com sucesso, a interface local envia a mensagem “ok” para a interface remota. Caso contrário, envia a mensagem “cerror”.

De maneira análoga, a interface remota recebe a mensagem enviada pela interface local, e executa uma tarefa. Neste teste, foi implementado o caso de receber a mensagem “ok” ou “cerror”. O código testado foi o seguinte:

```
If strData = "ok" Then
   ok.Show 1, Me
 End If
 If strData = "cerror" Then
   ERROR_FRM.Show 1, Me
 End If
```

Caso a mensagem recebida (“strData”) seja “ok”, então a interface remota exibe ao operador remoto uma mensagem de confirmação da conexão. Da mesma forma, se a mensagem for “cerror”, a interface exibe uma mensagem de erro.

Este teste verificou a validade de adotar o mesmo princípio de comunicação (*chat*) entre operadores para as interfaces no caso da teleoperação.

No caso da teleoperação, o operador envia apenas um comando de cada vez (já que os comandos são inseridos por botões na interface, e é impossível apertar dois botões ao mesmo tempo), entretanto, para o caso da monitoração remota, é diferente, pois dois ou mais sensores podem ser ativados ao mesmo tempo. Seguindo o princípio apresentado para a teleoperação, supondo que os sensores “s1” e “s2” sejam ativados ao mesmo tempo (Figura 55), a interface local deve enviar à interface remota duas mensagens simultaneamente, uma para indicar que “s1” foi ativado e outra para indicar que “s2” foi ativado. No entanto, a ferramenta *sockets* não permite que diversas mensagens sejam enviadas ao mesmo tempo, e o resultado acaba sendo errôneo, ou seja, a interface remota pode receber apenas uma das duas mensagens ou nenhuma delas, exibindo um estado equivocado para o operador.

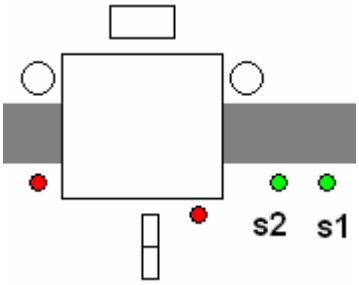

Figura 55 – Teste de monitoração remota.

Para contornar este problema, deve-se garantir que apenas uma única mensagem seja enviada de cada vez. Para isto, foi criado um vetor “estados” que armazena o estado de cada um dos 56 sensores existentes na interface. A cada intervalo de tempo determinado, todos os sensores são lidos e seus estados são armazenados no vetor. Em seguida, este vetor é enviado como mensagem para a interface remota, que ao recebê-lo, analisa o valor de cada elemento (em geral, “0” ou “1”) e reproduz os seus estados para o operador.

Os vetores e seus respectivos sensores estão mostrados nas Figuras 56, 57 e 58:

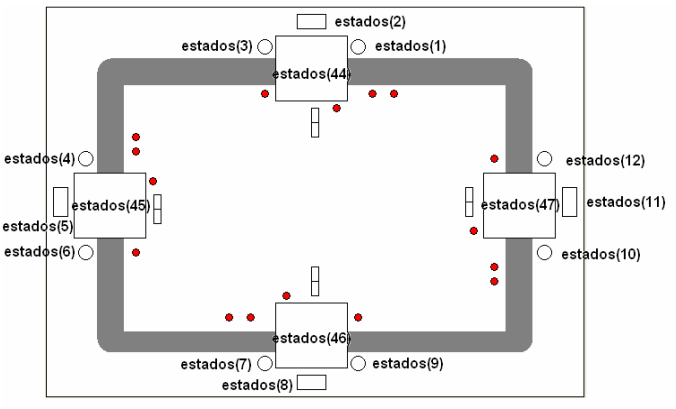

Figura 56 – Vetores dos atuadores da estação de transporte.

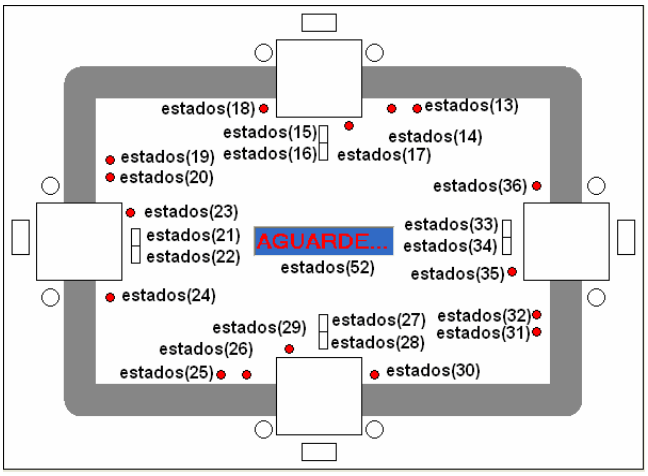

Figura 57 – Vetores dos sensores da estação de transporte.

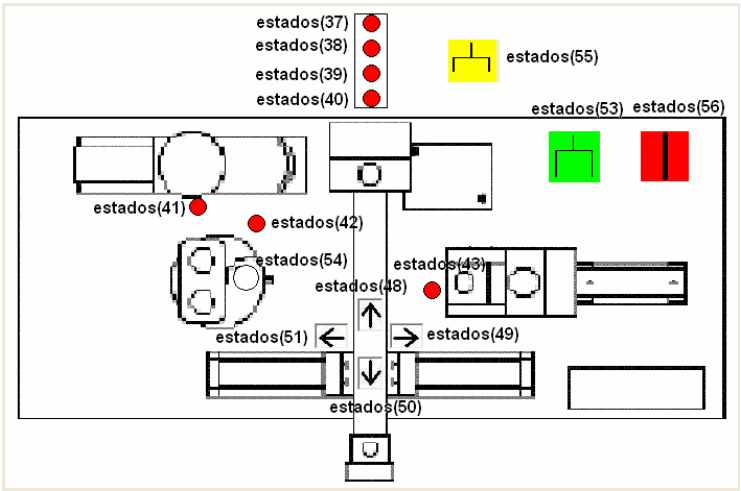

Figura 58 – Vetores dos sensores da estação de montagem.

É importante destacar que alguns dos sensores considerados de fato apenas indicam que o sinal foi enviado do CP ao atuador. Por exemplo, na interface da estação de transporte, verifica-se que os estados dos pistões também são representados (cor azul para estado “1” e cor branca para estado ‘0”), embora o sistema produtivo não contenha sensores para estes dispositivos. Da mesma forma, na estação de montagem, verifica-se que há indicadores de movimento (frente, trás, direita e esquerda) para os servomotores, embora não exista sensores para indicar tais movimentos. Os dispositivos que de fato são sensores estão representados nas interfaces pelas cores verde (estado “1”) e vermelho (estado “0”). A garra representada em amarelo na interface de montagem representa a posição aproximada da garra durante cada etapa do processo, e a lógica para sua representação é feita pela interface em VB, independente dos endereços do CP.

5.4) Teste das interfaces

Terminados os testes das ferramentas, as interfaces gráficas foram implementadas. O primeiro passo foi inserir os elementos representativos dos sensores, atuadores, carros, botões, etc, e em seguida programar os eventos e ações respectivas. Uma vez que a interface local foi implementada, o foco foi direcionado para a comunicação remota via *sockets*.

As interfaces implementadas são representadas nas Figuras 59 e 60.

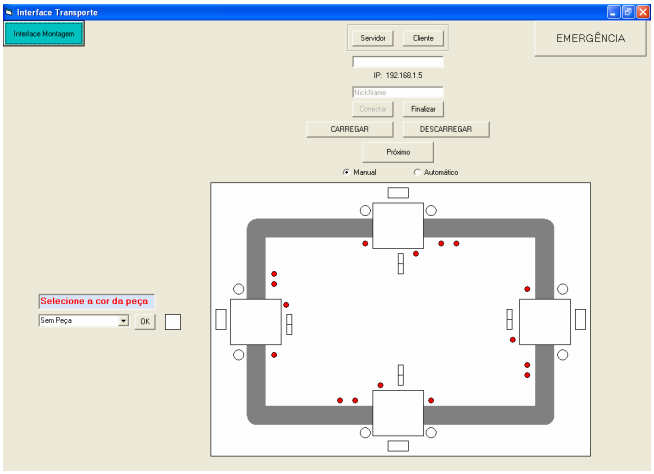

Figura 59 – Interface da estação de transporte.

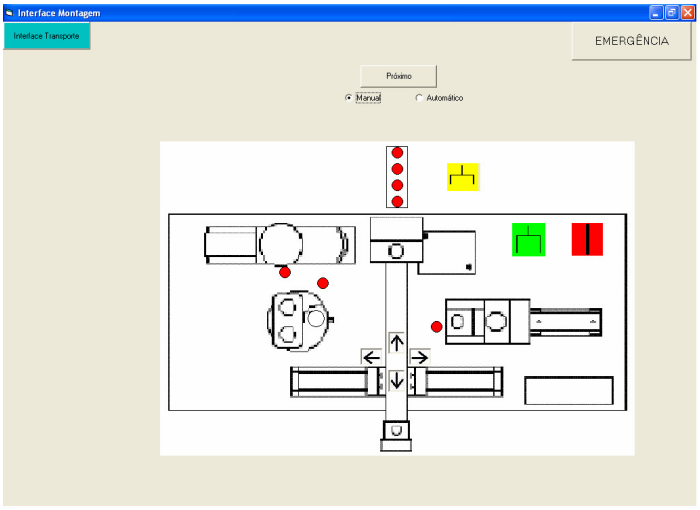

Figura 60 – Interface da estação de montagem.

As duas interfaces estão integradas e para alternar entre a interface de montagem e a de transporte basta clicar o botão azul localizado no canto superior esquerdo. Um operador remoto pode teleoperar o sistema pelos botões (próximo, iniciar, parar, emergência, etc) e também monitorá-lo pelos sensores incluídos graficamente nas interfaces.

Neste projeto, o programa de interface implementado pode ser utilizado tanto no computador local como no computador remoto. O próprio programa diferencia sua função (local ou remota) de acordo com a definição do operador, que no início do programa deve clicar no botão “Servidor” (operador local) ou no botão “Cliente” (operador remoto).

6) Conclusões

Os objetivos deste trabalho, ou seja, a implementação de interface gráfica de comunicação para supervisão de um sistema produtivo e a definição de um procedimento para especificação de como processar sinais de comando e monitoramento remoto de sistemas produtivos foram devidamente atingidos.

Seguem as principais dificuldades encontradas ao longo do desenvolvimento do projeto:

• A implementação do programa em *ladder* no STEP7 não é trivial e foi preciso freqüente contato com técnicos da empresa Siemens para dar continuidade ao trabalho.
• A implementação da interface remota em VB também não foi tarefa simples, pois na prática o número de sensores e atuadores envolvidos é relativamente elevado, o que exige um estudo prévio sobre estruturação do programa para lidar com um grande número de informações em um intervalo relativamente curto de tempo e com alta confiabilidade.

Em relação ao presente projeto considera-se que apesar de se ter alcançado plenamente os objetivos previstos, existem ainda outras partes para que um sistema de manufatura como um todo possa ser telecomandado e monitorado remotamente, isto é, o sistema aqui desenvolvido para as estações de transporte e de montagem deve ser revisto para que soluções similares sejam implementadas para as outras estações de trabalho. Além disso, é possível ainda introduzir dispositivos visuais como câmeras para monitorar o sistema e detectar outros tipos de erros rapidamente.

Especialmente no caso deste projeto, ressaltam-se a seguir alguns pontos de aprendizado prático:
• Em se tratando de qualquer máquina, dispositivo ou software é fundamental dispor de uma “boa” documentação e uma “boa” estratégia para a leitura desse material (e/ou dos arquivos de apoio). Isso contribui efetivamente para a

identificação tanto de procedimentos recomendáveis como de novas abordagens para os problemas aparentemente não previstos.

• Softwares que envolvem comunicação merecem cuidados especiais principalmente se houver mais de um tipo de comunicação deste software com o exterior. Isto é, a programação e funcionamento dependem de várias outras variáveis e assim cenários e situações específicas de teste e avaliação devem também ser considerados no desenvolvimento destes softwares. Por exemplo, no presente caso, tratar simultaneamente a comunicação Profibus concomitantemente com a comunicação via Internet (*sockets*) aumentou em muito o grau de dificuldade para o desenvolvimento do sistema.

• Um “bom” planejamento de como programas relativamente complexos devem ser desenvolvidos diminui as chances de erros no meio do processo. Pois sem isso pode-se perder o foco do trabalho e a desorganização das idéias certamente prejudica o produto final e compromete o cronograma.

• Durante um projeto extenso, documentar cada etapa importante de trabalho cumprido é fundamental para a tarefa de documentação final do projeto.

7) Referências bibliográficas

ALLCOMNET. Automação. Disponível em: <www.projetoweb.com.br/products/automation/clp.htm>. Acesso em Fevereiro de 2006.

ALMEIDA, Álvaro A. de. Introdução ao Visual Basic. 1998. Disponível em: <http://www.geocities.com/wallstreet/exchange/1726/computing/vb/vb_intro.htm> Acesso em Setembro de 2008.

ALTUS. Nota de aplicação P35. Disponível em: <http://www.altus.com.br/ftp/Public/Portugues/Notas%20de%20Aplicacao/NAP035%20-%20Exemplo%20Utilizacao%20Gateway%20PROFIBUS-DP_AS-I%20e%20Modulo%20Ponto%20AS-I%20IP67/NAP035.PDF>. Acesso em Setembro de 2008.

ÁLVARES, A.; SOUZA, A. Telemanufatura: Teleoperação via web da máquina de oxi-corte CNC autocut 2.5L. XVI Congresso Brasileiro de Engenharia Mecânica. 2001. Disponível em: <ftp://ftp.graco.unb.br/pub/publicacoes_graco/Cobem2001_TRB0897.pdf>. Acesso em Junho de 2008.

AXIS COMMUNICATIONS. Glossary: Network Vídeo. 2008. Disponível em: <http://www.axis.com/corporate/corp/glossary_video.htm#S>. Acesso em Junho de 2008.

BRASILIA VIRTUAL. Tudo sobre Visual Basic. 2008. Disponível em: <http://brasiliavirtual.info/tudo-sobre/visual-basic/ >. Acesso em Setembro de 2008.

CHRIST, Rafael E. R.; FIGUEREDO, Marcos V. M.; BASSANI, Thiago; Dias,

João da Silva; NIEVOLA, Júlio C.. Sistema de monitoração remota de pacientes em tempo-real através da Internet do hospital. Disponível em: <http://www.sbis.org.br/cbis9/arquivos/603.pdf>. Acesso em Abril de 2008.

DYSON, Peter. Novell dicionário de redes. Rio de Janeiro: Campus, 1995.

ECOCÂMARA. Coleta seletiva. 2008. Disponível em: <http://www2.camara.gov.br/internet/programas/ecocamara/colseletiva.monitoram ent o.html>. Acesso em Junho de 2008

FERREIRA, Almir Pires Neto. Aplicações práticas. 2000. Disponível em: <http://www.dei.unicap.br/~almir/seminarios/2000.1/telemedicina/aplipraticas.htm >. Acesso em Abril de 2008.

GOULART, Christiane Peres. Proposta de um modelo de referência para planejamento e controle da produção em empresas virtuais. Tese de Mestrado. Escola de Engenharia de São Carlos da Universidade de São Paulo. 2000.

MARTINS Filho, Luiz de Siqueira. CIC180 – Controle de processos por computador. Disponível em: <www.decom.ufop.br/prof/luizm/cic180_1.doc>. Acesso em Abril de 2008.

MATSUSAKI, C. Modelagem de sistemas de controle distribuídos e colaborativos de sistemas produtivos. Tese de doutorado. Escola Politécnica da Universidade de São Paulo. 2004.

MIYAGI, P. E., Controle Programável - Fundamentos do Controle de Sistemas a Eventos Discretos. São Paulo: Editora Edgard Blücher, 1996.

MIYAGI, P. E; SOUIT, S.; RETZ, R. SILVA, G.; SCHWOELK, J.; HASSUI, A. Monitoração remota de máquinas CNC. 2007. Disponível em:

<http://www.usp.br/siicusp/15Siicusp/1971.pdf>. Acesso em Junho de 2008.

NUNES, L. R. Sockets em Java. 2008. Disponível em: <http://www.sumersoft.com/publicacoes/SocketsEmJAVA.pdf>. Acesso em Junho de 2008.

PINHEIRO, Bruno de Lima. Implementação de um ambiente para simulação distribuída de sistemas produtivos. 2006. Trabalho apresentado à Escola Politécnica da USP para obtenção do título de Engenheiro.

PROFIBUS. Site oficial da organização mundial de usuários de Comunicação Profibus. 2006. Disponível em: <www.profibus.org>. Acesso em Abril de 2008.

ROCHA, Luis Fernando. CSO e a importância da monitoração remota. 11 de agosto de 2003. Disponível em: <http://www.cert.br/docs/reportagens/2003/2003-08-11.html>. Acesso em Fevereiro de 2006.

SIEMENS (a). Tudo sobre AS-Interface. 2008.

SIEMENS (b). Prodave MPI - Manual. 2005. Disponível em: < http://www.bitman.ca/Prodav_e.pdf>. Acesso em Junho de 2008.

SIEMENS (c). FM354 Servo Drive Positioning Module. Manual. 2008.

SIEMENS (d). FM 353, FM 354, FM 357(-2) in conjunction with PC Adapter. 2008. Disponível em: <http://support.automation.siemens.com/WW/view/en/299182>. Acesso em Setembro de 2008.

SIEMENS (e). Product information. CP5613 A2. 2008. Disponível em:

<http://cache.automation.siemens.com/dnl/TQ5NjAxAAAA_23556294_HB/pi_cp5613a2_74.pdf >. Acesso em Setembro de 2008.

SOUZA, Exson Machado. Desenvolvimento de componentes para facilitar a monitoração remota de redes de computadores usando a ferramenta WebManager. 2002. Disponível em: <http://www.dsc.ufcg.edu.br/~copin/pesquisa/bancodissertacoes/2002/ExsonMachadoSouza.pdf>. Acesso em Abril de 2008.

THE JAVA™ TUTORIAL. Tutorial explicativo sobre a ferramenta sockets. 2006. Disponível em: <http://java.sun.com/docs/books/tutorial/networking/sockets/index.html>. Acesso em Junho de 2008.

VILLANI, Emília. Modelagem e análise de sistemas supervisórios híbridos. Tese de doutorado. Escola Politécnica da Universidade de São Paulo. 2003.

VILLANI, E.+, CASTRO R. A.*, MARQUES F.M.*, Miyagi, P.E.*. +Instituto Tecnológico de Aeronáutica. *Escola Politécnica da Universidade de São Paulo. Remote monitoring and control of manufacturing system. Paper. 2006.

WIKIPEDIA (a). Controlador Lógico Programável. 2006. Disponível em: <http://pt.wikipedia.org/wiki/Controlador_l%C3%B3gico_program%C3%A1vel>. Acesso em Setembro de 2008

WIKIPEDIA (b). Servomotor. Disponível em: <http://pt.wikipedia.org/wiki/Servomotor>. Acesso em Setembro de 2008.

Whatis?.com. What is OCX?. 2001. Disponível em: <http://searchwinit.techtarget.com/sDefinition/0,,sid1_gci212689,00.html>. Acesso em Setembro de 2008.

Anexo A

Datasheet FM354

Este *Datasheet* contém os endereços e as respectivas funções do módulo FM354 para a programação dos servomotores via *software* STEP7.

AW-DB	FM 354							
Byte	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
General addresses								
DBW0	Module address (data type INT)							
DBW2 to DBB12	Reserved							
DBB13	Position	Execution started						
Control signals								
DBB14					Acknowl-edge opera-tor/travers-ing error		Switch to P bus Start-up	
DBB15	Drive enable	Block skip	Read-in enable	Acknow-ledge M function	Positive direction	Negative direction	Stop	Start
DBB16	Operating mode							
DBB17	Operating mode parameters							
DBB18	Override							
DBB19 to DBB21	Reserved							
Checkback signals								
DBB22	Channel initialized			Data error	Operator/ traversing error		Switch to P bus com-pleted	
DBB23		Reverse prog. scan	Dwell in progress			Wait for external enable	Machining in progress	Start enable
DBB24	Active operating mode							
DBB25	Position reached. Stop.		On-the-fly setting of actual value completed	Servo ena-ble status	Positive travel	Negative travel	End of measure-ment	Channel synchroni-zed
DBB26	M function number							
DBB27				M function modifica-tion				

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DBB28 to DBB33	Reserved							
Initiation signals								
Initiation signals for single settings (switches); transfer through Write request when change occurs								
DBB34	Simulation	Parking axis					On-the-fly measuring	Controller enable
DBB35	Automat. drift compensation disabled	Software limit positions disabled	Follow-up mode	Enable input disabled	Retrigger ref. point	Length measurement		
Initiation signals for single commands; transfer through Write request when change occurs (signals are reset following transfer)								
DBB36	Reserved							
DBB37		Rescind setting of actual value	Restart		Automatic block return	Automatic block advance	Delete residual distance	Activate MD
Initiation signals for Write requests								
DBB38	Set actual value	Set reference point		On-the-fly MDI block	MDI block	Setpoint for incremental dimension	Voltage levels 1, 2	Speed levels 1, 2
DBB39	Teach-in	Request application data	Program selection	Digital outputs	Modify parameters / data		Zero offset	On-the-fly setting of actual value
DBB40 to DBB41	Reserved							
Initiation signals for Read requests								
DBB42			Operating error no.	Service data	Actual value block change	Next NC block	Active NC block	Basic operating data
DBB43	Read measured values	Application data	Additional operating data	Dig. inputs/ outputs	Parameter/ data			

Ready signals								
Status/checkback signals from FC POS_CTRL								
DBB44	Simulation	Parking axis					On-the-fly measuring	Controller enable

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DBB45	Automat. drift compensation disabled	Software limit positions disabled	Follow-up mode	Enable input disabled	Retrig. ref. point	Length measurement		
DBB46	Reserved							
DBB47		Rescind setting of actual value	Restart		Autom. block return	Autom. block advance	Delete residual distance	Activate MD
DBB48	Set actual value	Set reference point		On-the-fly MDI block	MDI block	Setpoint for incremental dimension	Voltage levels 1, 2	Speed levels 1, 2
DBB49	Teach-in	Request application data	Program selection	Digital outputs	Modify parameters/ data		Zero offset	On-the-fly setting of actual value
DBB50 to DBB51	Reserved							
DBB52	Data error read	Operator/ traversing error read	Operating error read	Service data	Actual value block change	Next NC block	Active NC block	Basic operating data
DBB53	Read measured values	Application data	Additional operating data	Dig. inputs/ outputs	Parameter/ data			

Error signals

Error messages from FC POS_CTRL								
DBB54	Simulation	Parking axis					On-the-fly measuring	Controller enable
DBB55	Automat. drift compensation disabled	Software limit positions disabled	Follow-up mode	Enable input disabled	Retrig. ref. point	Length measurement		
DBB56	Reserved							
DBB57		Rescind setting of actual value	Restart		Autom. block return	Autom. block advance	Delete residual distance	Activate MD
DBB58	Set actual value	Set reference point		On-the-fly MDI block	MDI block	Setpoint for incremental dimension	Voltage levels 1, 2	Speed levels 1, 2
DBB59	Teach-in	Request application data	Program selection	Digital outputs	Modify parameters/ data		Zero offset	On-the-fly setting of actual value

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DBB60 to DBB61	Reserved							
DBB62	Data error read	Operator/ traversing error read	Operating error read	Service data	Actual value block change	Next NC block	Active NC block	Basic operating data
DBB63	Read measured values	Application data	Additional operating data	Dig. inputs/ outputs	Parameter/ data			
DBB64 bis DBB65	Reserved							

Processing status of FC POS_CTRL

DBW66	Error code (communications error) of the last job request/transfer (data type: INT)							
DBB68					Read request not possible	Read job active	Write request not possible	Write job active
DBB69							Reset status/error	

Diagnostic data for the FM, read out with FC POS_DIAG								
DBB70		Module not initialized			Ext. chan. err. (DBB78)	External error	Int./HW err. (DBB 72, 73)	Module/ group fault
DBB71				Channel info available	Module type classes (08H)			
DBB72				Int. module supply volt. failed	Watchdog triggered		Comm. error (K bus)	
DBB73		Process int. lost			RAM error	FEPROM error		
DBB74	FM pos. ID (74H)							
DBB75	Length of diagnostic information (16)							
DBB76	Number of channels (1)							
DBB77								Channel error vector
DBB78	Operating error				Voltage mon. sensor	Error: no increm. or zero mark	Error: abs. encoder	Cable break (inc. encoder)
DBB79 to DBB85	Reserved							

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Error code following flagging of an "Operating error" (is read if operating error is set following FC POS_DIAG)								
DBB86	Error number (DS 164) – Detail event class							
DBB87	Error number (DS 164) – Detail event number							
DBB88 to DBB89	Reserved							
Error code following flagging of "Operator-/traversing error"								
DBB90	Error number (DS 162) – Detail event class							
DBB91	Error number (DS 162) – Detail event number							
DBB92 to DBB93	Reserved							

Error code following flagging of "Data error"	
DBB94	Error number (DS163) – Detail event class
DBB95	Error number (DS163) – Detail event number
DBW96	Error code FC POS_DIAG (Return code SFC 51) (Data type: INT)
DBW98	Error code FC POS_MSRM (Return code SFC 59) (Data type: INT))
Data for the requests	
Zero offset	
DBD140	Data type: DINT
Set actual value	
DBD144	Data type: DINT
On-the-fly setting of actual value	
DBD148	Data type: DINT
Set reference point	
DBD152	Data type: DINT
Setpoint for incremental dimension	
DBD156	
Speed levels 1 and 2	
DBD160	Speed level 1
DBD164	Speed level 2
Voltage levels 1 and 2	
DBD168	Voltage level 1
DBD172	Voltage level 2

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MDI block								
DBB176 to DBB177	Reserved							
DBB178				Position/ dwell			G function group 2	G function group 1
DBB179					M function group 3	M function group 2	M function group 1	Speed
DBB180	G function no. of group 1							
DBB181	G function no. of group 2							
DBB182 bis DBB183	Reserved							
DBD184	Value for position/dwell (data type DINT)							
DBD188	Value for speed (data type DINT)							
DBB192	M function no. of group 1							
DBB193	M function no. of group 2							
DBB194	M function no. of group 3							
DBB195	Reserved							
Modify parameter/data or request reading of relevant data								
DBB196	DB type							
DBB197	Number							
DBB198	Quantity							
DBB199	Request							
DBB200 to DBB219	Data array, structure/data type of Write data as per bytes 1 to 4 of this structure (e.g. a program block or max. 5 MD items)							
Digital inputs/outputs								
DBB220					Digital input 3	Digital input 2	Digital input 1	Digital input 0
DBB221					Digital output 3	Digital output 2	Digital output 1	Digital output 0
On-the-fly MDI block								
DBB222 to DBB223	Reserved							
DBB224				Position/ dwell			G function group 2	G function group 1
DBB225					M function group 3	M function group 2	M function group 1	Speed
DBB226	G function no. of group 1							

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DBB227	G function no. of group 2							
DBB228 to DBB229	Reserved							
DBD230	Value for position/dwell (data type DINT)							
DBD234	Value for speed (data type DINT)							
DBB238	M function no. of group 1							
DBB239	M function no. of group 2							
DBB240	M function no. of group 3							
DBB241	Reserved							
Program selection								
DBB242	Program number							
DBB243	Block number							
DBB244	Direction of processing							
DBB245	Reserved							
Request for application data								
DBB246	Application data 1							
DBB247	Application data 2							
DBB248	Application data 3							
DBB249	Application data 4							
Teach-in								
DBB250	Program number							
DBB251	Block number							
DBB252 to DBB309	Reserved							

Data read as per request	
Basic operating data	
DBD310	Actual position (data type DINT)
DBD314	Actual speed
DBD318	Residual distance (data type DINT)
DBD322	Setpoint position (data type DINT)
DBD326	Sum of active coordinate offset, tool offset, zero offset (data type DINT)
DBD330	Rotational speed
DBD334 to DBD338	Reserved
Active NC block	
DBB342	Program number

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DBB343	Block number							
DBB344	Block skip	UP call	No. of UP calls	Position/ dwell		G function group 3	2	1
DBB345				Tool offset	M function group 3	2	1	Speed
DBB346	G function no. of group 1							
DBB347	G function no. of group 2							
DBB348	G function no. of group 3							
DBB349	Reserved							
DBD350	Value for position/dwell (data type DINT)							
DBD354	Value for speed (data type DINT)							
DBB358	M function no. of group 1							
DBB359	M function no. of group 2							
DBB360	M function no. of group 3							
DBB361	Tool offset no.							
Next NC block								
DBB362	Program number							
DBB363	Block number							
DBB364	Block skip	UP call	No. of UP calls	Position/ dwell		G function group 3	2	1
DBB365				Tool offset	M function group 3	2	1	Speed
DBB366	G function no. of group 1							
DBB367	G function no. of group 2							
DBB368	G function no. of group 3							
DBB369	Reserved							
DBD370	Value for position/dwell (data type DINT)							
DBD374	Value for speed (data type DINT)							
DBB378	M function no. of group 1							
DBB379	M function no. of group 2							
DBB380	M function no. of group 3							
DBB381	Tool offset no.							
Application data								
DBD382	Application data 1 (data type: DINT)							
DBD386	Application data 2 (data type: DINT)							
DBD390	Application data 3 (data type: DINT)							
DBD394	Application data 4 (data type: DINT)							

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Actual value block change								
DBD398	Data type DINT							
Service data								
DBD402	DAO output value (data type DINT)							
DBD406	Actual encoder value (data type DINT)							
DBD410	Missing pulses (data type DINT)							
DBD414	K_v factor (data type DINT)							
DBD418	Following error (data type DINT)							
DBD422	Following error limit (data type DINT)							
DBD426	Setpoint overshoot value/switch adjustment (data type DINT)							
DBD430	Approach time/response time constant (data type DINT)							
Additional operating data								
DBB434	Override							
DBB435	NC traversing program no.							
DBB436	NC block no.							
DBB437	UP call counter							
DBB438	Active G90/91							
DBB439	Active G60/64							
DBB440	Active G43/44							
DBB441	Active D number							
DBB442					Accelera-tion/decel-eration limit	Limited to ± 10 V	Speed limit	
DBB443 to DBB445	Reserved							
Parameter/data								
DBB446	DB type (MD, incremental dimension or traversing program)							
DBB447	Number							
DBB448	Quantity							
DBB449	Job							
DBB450 to DBB469	Array, structure/data type according to data, to be read as per bytes 1 to 4 of this structure (e.g. a program record or max. 5 MD items)							
DBB470 to DBB485	Reserved							

AW-DB	FM 354							
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Measured values								
Measured values as per FC POS_MSRM call								
DBD486	Initial value or on-the-fly measured value (data type DINT)							
DBD490	Final value (data type DINT)							
DBD494	Measured length value							
Array for operator control/monitoring								
Operator control and monitoring								
DBB498	Volt. levels transferred	Speed levels transferred	Increm. dim. trans-ferred	Teach-in transferred	Prog. sel. transferred	MDI block transferred	Read MD	Write MD
DBB499	Operator/ traversing error	Data error	Diagnostic interrupt			Zero offset transferred	Set actual value transferred	MDI block transferred on-the-fly
DBW500	MD number							
DBD502	MD value (data type DINT)							
DBB506	Incremental dimension number							
DBB507	Reserved							
DBW508	Display number							
DBW510	Keyboard code							
DBW512	Reserved							
Operating mode selection								
DBB514		Jog mode	Auto mode	Auto/single block mode	MDI	Incremen-tal mode (relative)	Approach to refer-ence point	Open-loop control mode
DBB515	Acknowl-edge diagnostic interrupt	Acknowl-edge error						

Anexo B

Programa em STEP7

A seguir é ilustrado o programa implementado em STEP7 no CP.

```
- Leitura da rede ASI
CALL  "CP342-2Read"
      BG_Adr:=368
      DBNr  :=1
      DBAdr :=0
     NOP   0

- Escrita da rede ASI
  CALL  "CP342-2Write"
      DBNr  :=2
      DBadr :=0
      BG_Adr:=368
     NOP   0

- Configuração inicial do sistema
    AN    M      0.0
     A     M      2.0
     =     M    100.1
    A     M    100.1
     L     S5T#30S
     SP    T      1
     NOP   0
     NOP   0
     NOP   0
     A     T      1
     =     M    100.0
     A     M    100.0
     =     DB1.DBX    3.7

- Lógica Pistão 1
     AN    M      0.0
     A(
     ON    M      1.0
     O     M    100.5
     )
     =     DB2.DBX    6.6
     A     DB2.DBX    6.6
     =     DB1.DBX    1.0
     A     M      5.0
     AN    M      6.0
     AN    DB1.DBX    5.4
     A     DB1.DBX    5.6
     AN    DB1.DBX    5.7
     AN    DB1.DBX    5.1
     =     M    100.4
     A     M    100.4
     L     S5T#1S
     SD    T      2
     NOP   0
     NOP   0
     NOP   0
     A     T      2
     =     M    100.5

- Lógica Pistão 2
     AN    M      0.0
     A(
     ON    M      1.0
     O
     A     M      6.0
     AN    M      7.0
     )
     =     DB2.DBX    6.5
     AN    DB2.DBX    6.5
     =     DB1.DBX    1.1

- Lógica Pistão 3
     AN    M      0.0
     A(
     ON    M      1.0
     O     M    100.0
     O     M    100.7
     )
     =     DB2.DBX    6.4
     A     DB2.DBX    6.4
     =     DB1.DBX    1.2
     A     M      8.0
     =     M    100.6
     A     M    100.6
     L     S5T#1S500MS
     SP    T      3
     NOP   0
     NOP   0
     NOP   0
     A     T      3
     =     M    100.7

- Lógica Pistão 4
     AN    M      0.0
     A(
     ON    M      1.0
     O     M    100.0
     O     M    101.1
     )
     =     DB2.DBX    1.2
     A     DB2.DBX    1.2
     =     DB1.DBX    1.3
     A     M      5.0
     AN    M      6.0
     AN    DB1.DBX    0.0
     A     DB1.DBX    0.2
     AN    DB1.DBX    0.3
     AN    DB1.DBX    1.5
     =     M    101.0
     A     M    101.0
     L     S5T#1S
     SD    T      4
     NOP   0
     NOP   0
     NOP   0
```

```
    A    T     4
    =    M   101.1

- Lógica Pistão 5
    AN   M     0.0
    A(
    ON   M     1.0
    O
    A    M     6.0
    AN   M     7.0
    )
    =    DB2.DBX   1.1
    AN   DB2.DBX   1.1
    =    DB1.DBX   1.6

- Lógica Pistão 6
    AN   M     0.0
    A(
    ON   M     1.0
    O    M   100.0
    O    M   100.7
    )
    =    DB2.DBX   1.0
    A    DB2.DBX   1.0
    =    DB1.DBX   1.7

- Lógica Pistão 7
    AN   M     0.0
    A(
    ON   M     1.0
    O    M   100.0
    O    M   101.3
    )
    =    DB2.DBX   3.6
    A    DB2.DBX   3.6
    =    DB1.DBX   4.0
    A    M     5.0
    AN   M     6.0
    AN   DB1.DBX   2.4
    A    DB1.DBX   2.6
    AN   DB1.DBX   2.7
    AN   DB1.DBX   2.1
    =    M   101.2
    A    M   101.2
    L    S5T#1S
    SD   T     5
    NOP  0
    NOP  0
    NOP  0
    A    T     5
    =    M   101.3

- Lógica Pistão 8
    AN   M     0.0
    A(
    ON   M     1.0
    O
    A    M     6.0
    AN   M     7.0
    )
    =    DB2.DBX   3.5
    AN   DB2.DBX   3.5
    =    DB1.DBX   4.1

- Lógica Pistão 9
    AN   M     0.0
    A(
    ON   M     1.0
    O    M   100.0
    O    M   100.7
    )
    =    DB2.DBX   3.4
    A    DB2.DBX   3.4
    =    DB1.DBX   4.2


- Lógica Pistão 10
    AN   M     0.0
    A(
    ON   M     1.0
    O    M   100.0
    O    M   101.5
    )
    =    DB2.DBX   4.2
    A    DB2.DBX   4.2
    =    DB1.DBX   4.3
    A    M     5.0
    AN   M     6.0
    AN   DB1.DBX   3.0
    A    DB1.DBX   3.2
    AN   DB1.DBX   3.3
    AN   DB1.DBX   4.5
    =    M   101.4
    A    M   101.4
    L    S5T#1S
    SD   T     6
    NOP  0
    NOP  0
    NOP  0
    A    T     6
    =    M   101.5

- Lógica Pistão 11
    AN   M     0.0
    A(
    ON   M     1.0
    O
    A    M     6.0
    AN   M     7.0
    )
    =    DB2.DBX   4.1
    AN   DB2.DBX   4.1
    =    DB1.DBX   4.6

- Lógica Pistão 12
    AN   M     0.0
    A(
    ON   M     1.0
    O    M   100.0
    O    M   100.7
    )
    =    DB2.DBX   4.0
    A    DB2.DBX   4.0
    =    DB1.DBX   4.7

- Configuração do servomotor 1
    CALL  FC    4
     DB_NO  :=30
     RET_VAL:=MW200
    NOP  0

- Configuração do servomotor 2
    CALL  FC    4
     DB_NO  :=40
     RET_VAL:=MW201
    NOP  0

- Operações servomotor 1
    A    M     1.2
    =    L    20.0
```

```
    A    L    20.0
    JNB  _001
    L    L#3
    T    DB30.DBB  16
_001: NOP  0
    A    L    20.0
    JNB  _002
    L    L#100
    T    DB30.DBB  18
_002: NOP  0
    A    L    20.0
    BLD  102
    =    DB30.DBX  34.0
    A    L    20.0
    AN   M    0.0
    =    DB30.DBX  15.7
    A    M    1.3
    FP   M    107.0
    =    M    107.1
    A    M    1.4
    =    L    20.0
    A    L    20.0
    BLD  102
    =    DB30.DBX  15.5
    A    L    20.0
    AN   M    1.7
    =    DB30.DBX  39.5
    A    L    20.0
    JNB  _003
    L    L#8
    T    DB30.DBB  16
_003: NOP  0
    A    L    20.0
    JNB  _004
    L    L#100
    T    DB30.DBB  18
_004: NOP  0
    A    L    20.0
    AN   M    1.6
    JNB  _005
    L    L#10
    T    DBB  242
_005: NOP  0
    A    L    20.0
    JNB  _006
    L    L#0
    T    DBB  243
_006: NOP  0
    A    M    1.5
    FP   M    107.2
    =    M    107.3
    A    M    1.6
    JNB  _007
    L    L#1
    T    DBB  242
_007: NOP  0
    A    M    12.0
    FP   M    104.0
    =    M    104.1
    A    M    16.0
    FP   M    104.2
    =    M    104.3
    A    M    20.0
    FP   M    104.4
    =    M    104.5
    A    M    26.0
    FP   M    104.6
    =    M    104.7
    A    M    30.0
    FP   M    105.0
    =    M    105.1
    A    M    34.0
    FP   M    105.2
    =    M    105.3
    A    M    39.0
    FP   M    105.4
    =    M    105.5
    A    M    44.0
    FP   M    105.6
    =    M    105.7
    A    M    50.0
    FP   M    106.0
    =    M    106.1
    A    M    56.0
    FP   M    106.2
    =    M    106.3
    A    M    60.0
    FP   M    106.4
    =    M    106.5
    A    M    64.0
    FP   M    106.6
    =    M    106.7
    O    M    104.1
    O    M    104.3
    O    M    104.5
    O    M    104.7
    O    M    105.1
    O    M    105.3
    O    M    105.5
    O    M    105.7
    O    M    106.1
    O    M    106.3
    O    M    106.5
    O    M    106.7
    O    M    107.1
    O    M    107.3
    =    DB30.DBX  15.0
    A    DB30.DBX  25.2
    =    DB1.DBX   2.3
    A    DB30.DBX  25.3
    =    DB1.DBX   5.3
```

- Operações servomotor 2

```
 A    M    1.2
    =    L    20.0
    A    L    20.0
    JNB  _008
    L    L#3
    T    DB40.DBB  16
_008: NOP  0
    A    L    20.0
    JNB  _009
    L    L#50
    T    DB40.DBB  18
_009: NOP  0
    A    L    20.0
    BLD  102
    =    DB40.DBX  34.0
    A    L    20.0
    AN   M    0.0
    =    DB40.DBX  15.7
 A    M    1.4
    =    L    20.0
    A    L    20.0
    BLD  102
    =    DB40.DBX  15.5
    A    L    20.0
    AN   M    1.7
    =    DB40.DBX  39.5
    A    L    20.0
```

```
    JNB   _00a
    L     L#8
    T     DB40.DBB   16
_00a: NOP   0
    A     L     20.0
    JNB   _00b
    L     L#100
    T     DB40.DBB   18
_00b: NOP   0
    A     L     20.0
    AN    M     1.6
    JNB   _00c
    L     L#10
    T     DBB  242
_00c: NOP   0
    A     L     20.0
    JNB   _00d
    L     L#0
    T     DBB  243
_00d: NOP   0
    A     M     1.6
    JNB   _00e
    L     L#1
    T     DBB  242
_00e: NOP   0
 O    M    104.1
    O     M    104.3
    O     M    104.5
    O     M    104.7
    O     M    105.1
    O     M    105.3
    O     M    105.5
    O     M    105.7
    O     M    106.1
    O     M    106.3
    O     M    106.5
    O     M    106.7
    O     M    107.1
    O     M    107.3
    =     DB40.DBX   15.0
    A     DB40.DBX   25.2
    =     DB1.DBX    2.2
    A     DB40.DBX   25.3
    =     DB1.DBX    5.2

- Lógica garra Z-
AN    M      0.0
    A(
    A     M      9.0
    AN    M     11.0
    AN    I     16.5
    O
    A     M     13.0
    AN    M     15.0
    AN    I     16.7
    O
    A     M     17.0
    AN    M     19.0
    AN    I     16.7
    O
    A     M     21.0
    AN    I     16.7
    AN    M     24.0
    O
    A     M     27.0
    AN    I     16.6
    AN    M     29.0
    O
    A     M     31.0
    AN    I     16.7
    AN    M     33.0
    O
    A     M     36.0
    AN    I     16.7
    AN    M     38.0
    O
    A     M     40.0
    AN    I     16.7
    AN    M     42.0
    O
    A     M     47.0
    AN    I     16.7
    AN    M     49.0
    O
    A     M     52.0
    AN    I     16.7
    AN    M     55.0
    O
    A     M     57.0
    AN    I     16.7
    AN    M     59.0
    O
    A     M     61.0
    AN    I     16.7
    AN    M     63.0
    O
    A     M     65.0
    AN    I     16.5
    AN    M     67.0
    )
    =     M    102.1

- Lógica garra Z+
  AN    M      0.0
    A(
    A     M     11.0
    AN    I     16.4
    AN    M     13.0
    O
    A     M     15.0
    AN    I     16.6
    AN    M     17.0
    O
    A     M     19.0
    AN    I     16.6
    AN    M     21.0
    O
    A     M     24.0
    AN    I     16.5
    AN    M     27.0
    O
    A     M     29.0
    AN    I     16.4
    AN    M     31.0
    O
    A     M     33.0
    AN    I     16.6
    AN    M     36.0
    O
    A     M     38.0
    AN    I     16.6
    AN    M     40.0
    O
    A     M     42.0
    AN    I     16.6
    AN    M     47.0
    O
    A     M     49.0
    AN    I     16.6
    AN    M     52.0
```

```
    O
    A    M    55.0
    AN   I    16.6
    AN   M    57.0
    O
    A    M    59.0
    AN   I    16.6
    AN   M    61.0
    O
    A    M    63.0
    AN   I    16.4
    AN   M    65.0
    O
    A    M    67.0
    AN   I    16.4
    O
    A    M     1.1
    AN   I    16.4
    AN   M     1.2
    )
    =    M   102.3
    A    M   102.3
    =    Q    16.4
   A     M   102.1
    =    Q    16.5
```

- Mostra pino preto

```
 AN   M     0.0
    A(
    O    M     3.0
    O    M    30.0
    )
    AN   M    31.0
    =    Q    20.0
    =    M   109.1
   A     M   109.1
    =    DB1.DBX    3.6
```

- Mostra pino prata

```
  AN   M     0.0
    A(
    A    M     3.1
    AN   M    30.0
    O    M    31.0
    )
    =    Q    20.1
    =    M   109.2
    A    M   109.2
    =    DB1.DBX    3.5
```

- Pistão da mola

```
  AN   M     0.0
    A    M     1.1
    AN   M    35.0
    =    Q    20.2
```

- Pistão da tampa

```
 AN   M     0.0
    A(
    A    M     1.1
    AN   M    45.0
    O    M    46.0
    )
    =    Q    20.4
    AN   M     0.0
    A    M    45.0
    AN   M    46.0
    =    Q    20.5
```

- Prende/solta peça na base de montagem

```
    AN   M     0.0
    A(
    A    M     1.1
    AN   M    23.0
    O    M    54.0
    )
    =    Q    20.7
```

- Lógica abre/fecha garra

```
 AN   M     0.0
    A(
    A    M    10.0
    AN   M    14.0
    O
    A    M    18.0
    AN   M    22.0
    O
    A    M    28.0
    AN   M    32.0
    O
    A    M    37.0
    AN   M    41.0
    O
    A    M    48.0
    AN   M    58.0
    O
    A    M    62.0
    AN   M    66.0
    )
    =    M   102.0
    A    M   102.0
    =    Q    16.0
   AN    M   102.0
    =    Q    16.1
```

- Gira garra a 0 graus

```
  AN   M     0.0
    A(
    A    M    43.0
    AN   M    51.0
    O
    A    M     1.2
    AN   M    25.0
    O    M    53.0
    )
    =    Q    16.2
```

- Gira garra a 270 graus

```
 AN   M     0.0
    A(
    A    M    25.0
    AN   M    43.0
    O
    A    M    51.0
    AN   M    53.0
    )
    =    Q    16.3
```

- Acknowledge all

```
    A    M     4.0
    =    DB30.DBX   14.3
    =    DB40.DBX   14.3
```

Anexo C

Programa em VB

A seguir é ilustrado o programa implementado em VB no computador.

C1) Module 1

Este modulo é onde são feitas as declarações das variáveis, funções e bibliotecas usadas no programa.

```
' PRODAVE S7 declarations in Visual Basic
' for S7_300 or S7_200
'
' Projekt/project -> Eigenschaften/Propertys ->
erstellen/build
' S7_300 = 1    or    S7_200 = 1
'
'
'****************************************************************
**************************************************************
'declarations for S7-200/300/400
'****************************************************************
**************************************************************
Type infotyp
    plcsw As Integer
    pgsw As Integer
    mlfb As String * 30
End Type

Type mixdatatype
    type As Byte
    size As Byte
    bstno As Integer
    no As Integer
End Type

Type plcadrtype
    adr As Byte
    SEGMENTID As Byte
    SLOTNO As Byte
    RACKNO As Byte
End Type

Public Const BST_IN_RAM As Integer = 16
Public Const BST_IN_EPROM As Integer = 32

'****************************************************************
**************************************************************
'declarations for S7-300/400
'****************************************************************
**************************************************************
Declare Function GetModuleFileNameA Lib
"kernel32.dll" (ByVal hModule As Long, szPath As Byte,
ByVal szPathLen As Long) As Long

Declare Function load_tool Lib "w95_s7.dll" (ByVal nr
As Byte, ByVal dev As String, adr As plcadrtype) As
Long
Declare Function new_ss Lib "w95_s7.dll" (ByVal nr As
Byte) As Long
Declare Function unload_tool Lib "w95_s7.dll" () As
Long

Declare Function ag_info Lib "w95_s7.dll" (value As
infotyp) As Long
Declare Function ag_zustand Lib "w95_s7.dll" (value
As Byte) As Long

Declare Function db_buch Lib "w95_s7.dll" (value As
Integer) As Long

' blockno, no, amount, value
Declare Function db_read Lib "w95_s7.dll" (ByVal db
As Long, ByVal dw As Long, anz As Long, value As
Integer) As Long
Declare Function db_write Lib "w95_s7.dll" (ByVal db
As Long, ByVal dw As Long, anz As Long, value As
Integer) As Long
```

```
 Declare Function d_field_read Lib "w95_s7.dll" (ByVal
db As Long, ByVal nr As Long, ByVal anz As Long,
value As Byte) As Long
Declare Function d_field_write Lib "w95_s7.dll" (ByVal
db As Long, ByVal nr As Long, ByVal anz As Long,
value As Byte) As Long
' no, amount, value
Declare Function e_field_read Lib "w95_s7.dll" (ByVal
nr As Long, ByVal anz As Long, value As Byte) As
Long
Declare Function a_field_read Lib "w95_s7.dll" (ByVal
nr As Long, ByVal anz As Long, value As Byte) As
Long
Declare Function a_field_write Lib "w95_s7.dll" (ByVal
nr As Long, ByVal anz As Long, value As Byte) As
Long
Declare Function m_field_read Lib "w95_s7.dll" (ByVal
nr As Long, ByVal anz As Long, value As Byte) As
Long
Declare Function m_field_write Lib "w95_s7.dll" (ByVal
nr As Long, ByVal anz As Long, value As Byte) As
Long
Declare Function t_field_read Lib "w95_s7.dll" (ByVal
nr As Long, ByVal anz As Long, value As Integer) As
Long
Declare Function z_field_read Lib "w95_s7.dll" (ByVal
nr As Long, ByVal anz As Long, value As Integer) As
Long
Declare Function z_field_write Lib "w95_s7.dll" (ByVal
nr As Long, ByVal anz As Long, value As Integer) As
Long
' data, value
Declare Function mix_read Lib "w95_s7.dll" (data As
mixdatatype, value As Byte) As Long
Declare Function mix_write Lib "w95_s7.dll" (data As
mixdatatype, value As Byte) As Long
' no, bitno
Declare Function mb_setbit Lib "w95_s7.dll" (ByVal no
As Long, ByVal bitno As Long) As Long
Declare Function mb_resetbit Lib "w95_s7.dll" (ByVal
no As Long, ByVal bitno As Long) As Long
' no, bitno, value
Declare Function mb_bittest Lib "w95_s7.dll" (ByVal no
As Long, ByVal bitno As Long, value As Byte) As Long

'****************************************************************
**************************************************************

'declarations for TeleService
'****************************************************************
**************************************************************

'modem name, standort name, tel.no, username,
password , window handel, message, wparam
Declare Function ts_dial Lib "w95_s7.dll" (ByVal
ModemName As String, ByVal StandortName As
String, ByVal TelNo As String, ByVal UserName As
String, ByVal Password As String, ByVal Handle As
Long, ByVal Message As Long, ByVal WParam As
Integer, ByVal res1 As Long) As Long
Declare Function ts_hang_up_dial Lib "w95_s7.dll" ()
As Long

'set indicaton on ring -> message if ts-adapter is
connected
'modem name, no of rings, window handel, message,
wparam
Declare Function ts_set_ringindicator Lib "w95_s7.dll"
(ByVal ModemName As String, ByVal Rings As Long,
ByVal Handle As Long, ByVal Message As Long, ByVal
WParam As Integer, ByVal res1 As String) As Long

'information from ring connected ts-adapter
'userid (16 bytes) , asid (1 byte)
Declare Function ts_read_info Lib "w95_s7.dll" (value
As Byte, MPIAdr As Byte) As Long

Declare Function ts_hang_up_ring Lib "w95_s7.dll" ()
As Long

'modem no, modem name, modem name len
Declare Function ts_get_modem_name Lib
"w95_s7.dll" (ByVal no As Long, ByVal ModemName
As String, ModemNameLen As Long) As Long

'****************************************************************
**************************************************************

'****************************************************************
**************************************************************
'declarations for komfort functions
'****************************************************************
**************************************************************
'errorno, errortext
Declare Function error_message Lib "komfort.dll"
(ByVal nr As Long, ByVal value$) As Long
```

```
'buffer, amount bytes
Declare Sub swab_buffer Lib "komfort.dll" (value%,
ByVal anz As Long)
Declare Sub copy_buffer Lib "komfort.dll" (value%,
value%, ByVal anz As Long)

Declare Function kg_to_float Lib "komfort.dll" (kg As
Integer, s As Single) As Long
Declare Function float_to_kg Lib "komfort.dll" (s As
Single, kg As Integer) As Long

Declare Sub gp_to_float Lib "komfort.dll" (gp As Long, s
As Single)
Declare Sub float_to_gp Lib "komfort.dll" (s As Single,
gp As Integer)

Declare Function kf_integer Lib "komfort.dll" (kf As
Integer) As Integer

'****************************************************************
************************************************************
'variables for demoprogram
'****************************************************************
************************************************************
Global DataTypeIdx As Integer
Global OutputTypeIdx As Integer
Global VerbIdx As Integer

Global value_word(1024) As Integer

'variáveis para interface transporte
Global value_byte0(1) As Byte
Global value_byte1(1) As Byte
Global value_byte2(1) As Byte
Global value_byte3(1) As Byte
Global value_byte4(1) As Byte
Global value_byte5(1) As Byte

'variáveis para interface montagem
Global value_byte(1) As Byte
Global value_byteA(1) As Byte

Global value_byteM(1) As Byte 'byte para escrever
flags

Global plcadr(5) As plcadrtype
Global mixdata(17) As mixdatatype

Global ErrorText As String

Global res As Long
Global AMOUNT As Long
Global BLOCKNO As Long
Global no As Long

Global ModemName As String
Global Standort As String
Global TelNo As String
Global UserName As String
Global Password As String
Global RingActive As Byte
Global UserID(17) As Byte
Global MPIAdr As Byte

Global str As String

Global estados(1 To 56) As String
Global frase As String

Global flag As Integer
Global flag2 As Integer

Global contador_load As Integer

Global indicador As Integer
Global trava As Integer

Global CS As Integer   'variavel para Cliente/Servidor:
0=local, 1=servidor, 2=cliente
```

C2) CLientFRM:

Este form é responsável pelas operações do computador remoto. As principais tarefas são a troca de dados via *chat* entre operadores, leitura do vetor de estados dos sensores e atuadoes e envio de comandos à interface local.

```
Public Sub s_client()
Winsock.SendData "M" & msg.Text  'envia dados ao
servidor
msg.Text = ""
End Sub

Private Sub Form_Load()
```

```
t_c.Enabled = True
End Sub

Private Sub t_c_Timer()
Call s_client
End Sub

'=========================================
==================================
'======================Exibe informações se
conectado======================
Private Sub Winsock_Close()
   MainTXT.SelText = ""¨°*.º.*°¨ Desconectado do
Servidor ¨°*.º.*°¨"" & vbCrLf 'exibe texto se
desconectado do servidor
   senddataCMD.Enabled = False 'não permite enviar
dados se o servidor for desconectado
End Sub

'=========================================
==================================
'======================Recebe dados do
servidor============================
Private Sub winsock_DataArrival(ByVal bytesTotal As
Long)
Dim strData, strData2 As String 'armazena dados
enviados pelo cliente
Call Winsock.GetData(strData, vbString) 'pega dados
enviados pelo clente

strData2 = Left(strData, 1) 'salva a primeira letra da
mensagem
strData = Mid(strData, 2) 'salva o restante da
mensagem

If strData2 = "C" Then 'informa form para carregar
   nickSERVER.Caption = strData 'carrega username
do servidor
   Me.Show 'mostra form cliente
   Winsock.SendData "N" & nickCLIENT.Caption 'envia
username ao servidor
   ClientFRM.Caption = "Chat   [Bem vindo, " &
nickCLIENT.Caption & "!]" 'renomeia o form
   MainTXT.SelText = ""¨°*.º.*°¨ Connectado ao
Servidor ¨°*.º.*°¨"" & vbCrLf 'mostra que conexão foi
feita
End If

If strData2 = "T" Then MainTXT.SelText =
nickSERVER.Caption & ":     " & strData & vbCrLf
'adiciona dados do servidor no textbox

'recebendo comandos do servidor
If strData2 = "M" Then
  If strData = "ok" Then
    ok.Show 1, Me
    strData = ""
  End If
  If strData = "cerror" Then
    ERROR_FRM.Show 1, Me
    strData = ""
  End If
  If strData = "unload_ok" Then
    unload.Show 1, Me
    strData = ""
  End If

'Fazendo leitura dos estados
Dim letra As String
Dim bool As Integer
Dim j As Integer

j = 1

While (j < 57)
   letra = Mid(strData, j, 1)
   If (letra = "") Then
     bool = 7 'valor escolhido aleatoriamente
   Else
     bool = Val(letra)
   End If

If (bool <> 7) Then
If (j = 1 And bool = 0) Then
   Form1.p1.FillColor = vbCyan
End If
If (j = 1 And bool = 1) Then
   Form1.p1.FillColor = vbWhite
End If
If (j = 2 And bool = 0) Then
   Form1.p2.FillColor = vbCyan
End If
If (j = 2 And bool = 1) Then
   Form1.p2.FillColor = vbWhite
```

```
End If
If (j = 3 And bool = 0) Then
   Form1.p3.FillColor = vbCyan
End If
If (j = 3 And bool = 1) Then
   Form1.p3.FillColor = vbWhite
End If
If (j = 4 And bool = 0) Then
   Form1.p4.FillColor = vbCyan
End If
If (j = 4 And bool = 1) Then
   Form1.p4.FillColor = vbWhite
End If
If (j = 5 And bool = 0) Then
   Form1.p5.FillColor = vbCyan
End If
If (j = 5 And bool = 1) Then
   Form1.p5.FillColor = vbWhite
End If
If (j = 6 And bool = 0) Then
   Form1.p6.FillColor = vbCyan
End If
If (j = 6 And bool = 1) Then
   Form1.p6.FillColor = vbWhite
End If
If (j = 7 And bool = 0) Then
   Form1.p7.FillColor = vbCyan
End If
If (j = 7 And bool = 1) Then
   Form1.p7.FillColor = vbWhite
End If
If (j = 8 And bool = 0) Then
   Form1.p8.FillColor = vbCyan
End If
If (j = 8 And bool = 1) Then
   Form1.p8.FillColor = vbWhite
End If
If (j = 9 And bool = 0) Then
   Form1.p9.FillColor = vbCyan
End If
If (j = 9 And bool = 1) Then
   Form1.p9.FillColor = vbWhite
End If
If (j = 10 And bool = 0) Then
   Form1.p10.FillColor = vbCyan
End If
If (j = 10 And bool = 1) Then
   Form1.p10.FillColor = vbWhite
End If
If (j = 11 And bool = 0) Then
   Form1.p11.FillColor = vbCyan
End If
If (j = 11 And bool = 1) Then
   Form1.p11.FillColor = vbWhite
End If
If (j = 12 And bool = 0) Then
   Form1.p12.FillColor = vbCyan
End If
If (j = 12 And bool = 1) Then
   Form1.p12.FillColor = vbWhite
End If

If (j = 13 And bool = 1) Then
   Form1.s1a.FillColor = vbGreen
End If
If (j = 13 And bool = 0) Then
   Form1.s1a.FillColor = vbRed
End If
If (j = 14 And bool = 1) Then
   Form1.s2.FillColor = vbGreen
End If
If (j = 14 And bool = 0) Then
   Form1.s2.FillColor = vbRed
End If
If (j = 15 And bool = 1) Then
   Form1.s4.FillColor = vbGreen
End If
If (j = 15 And bool = 0) Then
   Form1.s4.FillColor = vbRed
End If
If (j = 16 And bool = 1) Then
   Form1.s5.FillColor = vbGreen
End If
If (j = 16 And bool = 0) Then
   Form1.s5.FillColor = vbRed
End If
If (j = 17 And bool = 1) Then
   Form1.s3.FillColor = vbGreen
End If
If (j = 17 And bool = 0) Then
   Form1.s3.FillColor = vbRed
End If
If (j = 18 And bool = 1) Then
   Form1.s6.FillColor = vbGreen
```

```
End If
If (j = 18 And bool = 0) Then
   Form1.s6.FillColor = vbRed
End If
If (j = 19 And bool = 1) Then
   Form1.s7.FillColor = vbGreen
End If
If (j = 19 And bool = 0) Then
   Form1.s7.FillColor = vbRed
End If
If (j = 20 And bool = 1) Then
   Form1.s8.FillColor = vbGreen
End If
If (j = 20 And bool = 0) Then
   Form1.s8.FillColor = vbRed
End If
If (j = 21 And bool = 1) Then
   Form1.s10.FillColor = vbGreen
End If
If (j = 21 And bool = 0) Then
   Form1.s10.FillColor = vbRed
End If
If (j = 22 And bool = 1) Then
   Form1.s11.FillColor = vbGreen
End If
If (j = 22 And bool = 0) Then
   Form1.s11.FillColor = vbRed
End If
If (j = 23 And bool = 1) Then
   Form1.s9.FillColor = vbGreen
End If
If (j = 23 And bool = 0) Then
   Form1.s9.FillColor = vbRed
End If
If (j = 24 And bool = 1) Then
   Form1.s12.FillColor = vbGreen
End If
If (j = 24 And bool = 0) Then
   Form1.s12.FillColor = vbRed
End If
If (j = 25 And bool = 1) Then
   Form1.s13.FillColor = vbGreen
End If
If (j = 25 And bool = 0) Then
   Form1.s13.FillColor = vbRed
End If
If (j = 26 And bool = 1) Then
   Form1.s14.FillColor = vbGreen
End If
If (j = 26 And bool = 0) Then
   Form1.s14.FillColor = vbRed
End If
If (j = 27 And bool = 1) Then
   Form1.s16.FillColor = vbGreen
End If
If (j = 27 And bool = 0) Then
   Form1.s16.FillColor = vbRed
End If
If (j = 28 And bool = 1) Then
   Form1.s17.FillColor = vbGreen
End If
If (j = 28 And bool = 0) Then
   Form1.s17.FillColor = vbRed
End If
If (j = 29 And bool = 1) Then
   Form1.s15.FillColor = vbGreen
End If
If (j = 29 And bool = 0) Then
   Form1.s15.FillColor = vbRed
End If
If (j = 30 And bool = 1) Then
   Form1.s18.FillColor = vbGreen
End If
If (j = 30 And bool = 0) Then
   Form1.s18.FillColor = vbRed
End If
If (j = 31 And bool = 1) Then
   Form1.s19.FillColor = vbGreen
End If
If (j = 31 And bool = 0) Then
   Form1.s19.FillColor = vbRed
End If
If (j = 32 And bool = 1) Then
   Form1.s20.FillColor = vbGreen
End If
If (j = 32 And bool = 0) Then
   Form1.s20.FillColor = vbRed
End If
If (j = 33 And bool = 1) Then
   Form1.s22.FillColor = vbGreen
End If
If (j = 33 And bool = 0) Then
   Form1.s22.FillColor = vbRed
End If
```

```
If (j = 34 And bool = 1) Then
   Form1.s23.FillColor = vbGreen
End If
If (j = 34 And bool = 0) Then
   Form1.s23.FillColor = vbRed
End If
If (j = 35 And bool = 1) Then
   Form1.s21.FillColor = vbGreen
End If
If (j = 35 And bool = 0) Then
   Form1.s21.FillColor = vbRed
End If
If (j = 36 And bool = 1) Then
   Form1.s24.FillColor = vbGreen
End If
If (j = 36 And bool = 0) Then
   Form1.s24.FillColor = vbRed
End If

If (j = 37 And bool = 1) Then
   Form2.b27.FillColor = vbGreen
End If
If (j = 37 And bool = 0) Then
   Form2.b27.FillColor = vbRed
End If
If (j = 38 And bool = 1) Then
   Form2.b26.FillColor = vbGreen
End If
If (j = 38 And bool = 0) Then
   Form2.b26.FillColor = vbRed
End If
If (j = 39 And bool = 1) Then
   Form2.b25.FillColor = vbGreen
End If
If (j = 39 And bool = 0) Then
   Form2.b25.FillColor = vbRed
End If
If (j = 40 And bool = 1) Then
   Form2.b24.FillColor = vbGreen
End If
If (j = 40 And bool = 0) Then
   Form2.b24.FillColor = vbRed
End If
If (j = 41 And bool = 1) Then
   Form2.b14.FillColor = vbGreen
End If
If (j = 41 And bool = 0) Then
   Form2.b14.FillColor = vbRed
End If
If (j = 42 And bool = 1) Then
   Form2.b15.FillColor = vbGreen
End If
If (j = 42 And bool = 0) Then
   Form2.b15.FillColor = vbRed
End If
If (j = 43 And bool = 1) Then
   Form2.b12.FillColor = vbGreen
End If
If (j = 43 And bool = 0) Then
   Form2.b12.FillColor = vbRed
End If

If (j = 44 And bool = 1) Then
   Form1.Shape1.FillColor = vbBlack
End If
If (j = 44 And bool = 0) Then
   Form1.Shape1.FillColor = vbWhite
End If
If (j = 45 And bool = 1) Then
   Form1.Shape2.FillColor = vbBlack
End If
If (j = 45 And bool = 0) Then
   Form1.Shape2.FillColor = vbWhite
End If
If (j = 46 And bool = 1) Then
   Form1.Shape3.FillColor = vbBlack
End If
If (j = 46 And bool = 0) Then
   Form1.Shape3.FillColor = vbWhite
End If
If (j = 47 And bool = 1) Then
   Form1.Shape4.FillColor = vbBlack
End If
If (j = 47 And bool = 0) Then
   Form1.Shape4.FillColor = vbWhite
End If
If (j = 48 And bool = 1) Then
   Form2.c1.Visible = True
   Form2.c0.Visible = False
End If
If (j = 48 And bool = 0) Then
   Form2.c0.Visible = True
   Form2.c1.Visible = False
End If
```

```
If (j = 49 And bool = 1) Then
   Form2.d1.Visible = True
   Form2.do.Visible = False
End If
If (j = 49 And bool = 0) Then
   Form2.do.Visible = True
   Form2.d1.Visible = False
End If
If (j = 50 And bool = 1) Then
   Form2.b1.Visible = True
   Form2.b0.Visible = False
End If
If (j = 50 And bool = 0) Then
   Form2.b0.Visible = True
   Form2.b1.Visible = False
End If
If (j = 51 And bool = 1) Then
   Form2.e1.Visible = True
   Form2.e0.Visible = False
End If
If (j = 51 And bool = 0) Then
   Form2.e0.Visible = True
   Form2.e1.Visible = False
End If
If (j = 52 And bool = 1) Then
   Form1.wait.Visible = True
End If
If (j = 52 And bool = 0) Then
   Form1.wait.Visible = False
End If
If (j = 53 And bool = 0) Then
   Form2.garraf.Visible = True
   Form2.garraa.Visible = False
End If
If (j = 53 And bool = 1) Then
   Form2.garraa.Visible = True
   Form2.garraf.Visible = False
End If
If (j = 54 And bool = 1) Then
   Form2.pino.FillColor = vbBlack
End If
If (j = 54 And bool = 0) Then
   Form2.pino.FillColor = &HC0C0C0
End If
If (j = 55 And bool = 1) Then
   Form2.g1.Visible = True
   Form2.g2.Visible = False
   Form2.g3.Visible = False
   Form2.g4.Visible = False
   Form2.g5.Visible = False
   Form2.g6.Visible = False
End If
If (j = 55 And bool = 2) Then
   Form2.g1.Visible = False
   Form2.g2.Visible = True
   Form2.g3.Visible = False
   Form2.g4.Visible = False
   Form2.g5.Visible = False
   Form2.g6.Visible = False
End If
If (j = 55 And bool = 3) Then
   Form2.g1.Visible = False
   Form2.g2.Visible = False
   Form2.g3.Visible = True
   Form2.g4.Visible = False
   Form2.g5.Visible = False
   Form2.g6.Visible = False
End If
If (j = 55 And bool = 4) Then
   Form2.g1.Visible = False
   Form2.g2.Visible = False
   Form2.g3.Visible = False
   Form2.g4.Visible = True
   Form2.g5.Visible = False
   Form2.g6.Visible = False
End If
If (j = 55 And bool = 5) Then
   Form2.g1.Visible = False
   Form2.g2.Visible = False
   Form2.g3.Visible = False
   Form2.g4.Visible = False
   Form2.g5.Visible = True
   Form2.g6.Visible = False
End If
If (j = 55 And bool = 6) Then
   Form2.g1.Visible = False
   Form2.g2.Visible = False
   Form2.g3.Visible = False
   Form2.g4.Visible = False
   Form2.g5.Visible = False
   Form2.g6.Visible = True
End If
If (j = 56 And bool = 1) Then
   Form2.garra270.Visible = True
```

```
   Form2.garra0.Visible = False
End If
If (j = 56 And bool = 0) Then
   Form2.garra270.Visible = False
   Form2.garra0.Visible = True
End If
End If 'if bool <> 7
j = j + 1
Wend
'FIM da leitura dos estados

End If  'if strData2 = "M"

End Sub
'=======================================
================================
'=======================Envia dados ao
servidor============================
Private Sub senddataCMD_Click()
MainTXT.SelText = nickCLIENT.Caption & ":     " &
dataTXT.Text & vbCrLf 'insere texto digitado
Winsock.SendData "T" & dataTXT.Text 'envia ao
servidor
dataTXT.Text = "" 'apaga o texto digitado
End Sub
'=======================================
================================
```

C3) ERROR_FRM:

Este form exibe uma mensagem de erro caso a conexão entre CP e computador local não seja estabelecida com sucesso.

```
Private Sub Form_Activate()
IDERROR.Text = "Erro de conexão!"
End Sub

Private Sub IDOK_Click()
ERROR_FRM.Visible = False
End Sub
```

C4) Form1:

Este é o form principal do programa, que representa a interface da estação de transporte e também onde se determina se a interface será local ou remota. Neste form são estabelecidas as conexões entre computadores local e remoto. São definidads também as tarefas de controle e monitoração, bem como a montagem do vetor de estados do programa inteiro.

```
Private Sub ClientCMD_Click()
IPtxt.Text = "IP do Servidor" 'pede para inserir ip do
servidor
Me.Caption = "Conectar como Cliente" 'renomeia
caption
IPtxt.Enabled = True 'botões
necessários/desnecessários para conectar
nicknameTXT.Enabled = True
ServerCMD.Enabled = False
ClientCMD.Enabled = False
ConnectCMD.Enabled = True
CS = 2   'a aplicação passa a ser cliente

End Sub

Function binstr2value(s As String) As Long
Dim v As Long
Dim w As Long
Dim i, j As Long

v = 0
w = 1

j = Len(s)
For i = 1 To j
   If (Mid$(s, i, 1) = "1") Then v = v Or w
   w = w * 2
Next i
binstr2value = v
End Function

Private Sub cmd1_Click()
 If (CS = 0) Then
   Call PROX
 End If 'CS = 0

 If (CS = 2) Then
   ClientFRM.msg.Text = "next"
 End If

End Sub
```

```
Private Sub PROX()
  Dim m As Integer
  'estação de transporte só movimenta se acabou
montagem
  If (flag2 = 8 And trava = 0) Then
    flag = flag + 1
    If (flag = 9) Then
      m = 5
      str = bin2.Text
      value_byteM(0) = binstr2value(str)
      While (m < 9)
        no = m
        res = m_field_write(no, AMOUNT,
value_byteM(0))
        m = m + 1
      Wend
      flag = 5 'O programa fica preso nos passos 5 a 8
    End If 'flag = 9
    txt_timer2.Text = flag 'TESTE
    If (flag = 6) Then
      Timer1.Enabled = False
      indicador = 0
    End If
    str = bin1.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    If (flag = 7) Then Timer1.Enabled = True
  End If 'flag2 = 8
End Sub

Private Sub cmd2_Click()
If (CS = 0) Then
  Timer2.Enabled = True
  cmd2.Enabled = False
  Option1.Enabled = False 'Garante que usuário não
mude para manual sem desligar o automático
End If

If (CS = 2) Then
  ClientFRM.msg.Text = "start"
  cmd2.Enabled = False
  Option1.Enabled = False 'Garante que usuário não
mude para manual sem desligar o automático
End If
End Sub

Private Sub cmd3_Click()
If (CS = 0) Then
  Timer2.Enabled = False
  cmd2.Enabled = True
  Option1.Enabled = True
End If

If (CS = 2) Then
  ClientFRM.msg.Text = "stop"
  cmd2.Enabled = True
  Option1.Enabled = True
End If
End Sub

Private Sub cmd4_Click()
Dim n As Integer

'local
If (CS = 0) Then
  plcadr(0).adr = 2
  plcadr(0).SEGMENTID = 0
  plcadr(0).RACKNO = 0
  plcadr(0).SLOTNO = 2
  plcadr(1).adr = 0
  VerbIdx = 0
  res = load_tool(1, "S7ONLINE", plcadr(0))

  If (res = 0) Then
    ok.Show 1, Me
    contador_load = 0
    Timer_load.Enabled = True
  End If 'res = 0
  If (res <> 0) Then ERROR_FRM.Show 1, Me
End If 'CS = 0

'cliente
If (CS = 2) Then
  ClientFRM.msg.Text = "load"
End If

End Sub

Private Sub cmd5_Click()
If (CS = 0) Then
  res = unload_tool()
  If (res = 0) Then unload.Show 1, Me
End If
```

```
If (CS = 2) Then ClientFRM.msg.Text = "unload"
End Sub

Private Sub cmd7_Click()
If (CS = 0) Then
'guarda cor escolhida
pega_cor.Text = cb1.Text

If cb1.Text = "Preto" Then
   peca.FillColor = vbBlack
   'altera cor do pino: 1 prata, 0 preto
   bin3.Text = "01000000"
   str = bin3.Text
   value_byteM(0) = binstr2value(str)
   no = 3
   res = m_field_write(no, AMOUNT, value_byteM(0))
End If
If cb1.Text = "Rosa" Then
   peca.FillColor = vbMagenta
   'altera cor do pino: 0 prata, 1 preto
   bin3.Text = "10000000"
   str = bin3.Text
   value_byteM(0) = binstr2value(str)
   no = 3
   res = m_field_write(no, AMOUNT, value_byteM(0))
End If
If cb1.Text = "Prata" Then
   peca.FillColor = &HC0C0C0
   'altera cor do pino: 0 prata, 1 preto
   bin3.Text = "10000000"
   str = bin3.Text
   value_byteM(0) = binstr2value(str)
   no = 3
   res = m_field_write(no, AMOUNT, value_byteM(0))
End If
If cb1.Text = "Sem Peça" Then
   peca.FillColor = vbWhite
End If
End If 'CS = 0

If (CS = 2) Then
  If (cb1.Text = "Preto") Then
    ClientFRM.msg.Text = "preto"
    peca.FillColor = vbBlack
  End If
  If (cb1.Text = "Rosa") Then
    ClientFRM.msg.Text = "rosa"
    peca.FillColor = vbMagenta
  End If
  If (cb1.Text = "Prata") Then
    ClientFRM.msg.Text = "prata"
    peca.FillColor = &HC0C0C0
  End If
  If (cb1.Text = "Sem Peça") Then
    ClientFRM.msg.Text = "sem_peca"
    peca.FillColor = vbWhite
  End If
End If 'CS = 2
End Sub

Private Sub ConnectCMD_Click()
If nicknameTXT.Text = "" Then Exit Sub 'não conecta
até inserir um Nickname

If Me.Caption = "Conectar como Servidor" Then
'conecta ao cliente
   ServerFRM.Winsock.Close 'fecha conexões
anteriores
   ServerFRM.Winsock.LocalPort = CLng(187) 'porta
187
   ServerFRM.Winsock.Listen 'escuta para ver se
cliente quer se conectar
   ServerFRM.nickSERVER.Caption =
nicknameTXT.Text 'coloca username para o server
End If

If Me.Caption = "Conectar como Cliente" Then 'conecta
ao servidor
   If IPtxt.Text = "IP do Servidor" Then Exit Sub 'caso
esqueça de colocar o IP do servidor
   If IPtxt.Text = "" Then Exit Sub
      ClientFRM.Winsock.Close 'fecha conexões
anteriores
      ClientFRM.Winsock.Connect IPtxt.Text, "187"
'envia IP para conectar ao servidor
      ClientFRM.nickCLIENT.Caption =
nicknameTXT.Text 'coloca username como cliente
End If

If (CS = 1) Then
'como servidor, suponho que haverá apenas
monitoração sem controle:
cmd1.Enabled = False
```

```
cmd2.Enabled = False
cmd3.Enabled = False
cmd4.Enabled = False
cmd5.Enabled = False

'como servidor, não preciso escolher a cor da peça
(isso é o cliente que faz)
cb1.Enabled = False
cmd7.Enabled = False

'como servidor, não é possível desconectar do cliente
disconnect.Enabled = False
End If 'CS=1

If (CS = 2) Then
'como cliente, não preciso do Prodave:
Form1.Timer3.Enabled = False
Form2.Timer3.Enabled = False

'como cliente, não preciso do timer1 e timer_trava
Timer1.Enabled = False
Timer_trava.Enabled = False
End If 'CS=2

'para todos os casos
ConnectCMD.Enabled = False

End Sub

Private Sub disconnect_Click()

If (CS = 2) Then
  ClientFRM.msg.Text = "finalizar"
End If

Timer_fim.Enabled = True

End Sub

Private Sub emergencia1_Click()
   If (CS = 0) Then
   no = 0
   str = bin1.Text
   value_byteM(0) = binstr2value(str)
   res = m_field_write(no, AMOUNT, value_byteM(0))
   Form1.Timer2.Enabled = False
   Form2.Timer2.Enabled = False
   emergencia1.BackColor = vbRed
   End If

   If (CS = 2) Then
     ClientFRM.msg.Text = "emergencia"
     emergencia1.BackColor = vbRed
   End If
End Sub

Private Sub Form_Load()

ConnectCMD.Enabled = False 'não conecta até inserir
os parâmetros necessários
YouripLBL.Caption = "IP:  " & Winsock.LocalIP 'exibe IP
IPtxt.Enabled = False 'desabilitado até selecionar entre
cliente ou servidor
nicknameTXT.Enabled = False
CS = 0  'por padrao o programa é apenas para
aplicação local

Form2.Visible = False

'inserindo elementos na Combobox
cb1.AddItem "Preto"
cb1.AddItem "Rosa"
cb1.AddItem "Prata"
cb1.AddItem "Sem Peça"

bin1.Text = "10000000"
bin2.Text = "00000000"
AMOUNT = 1 'no programa inteiro só vai ser exibido
um byte por vez

contador_load = 0
trava = 0

End Sub

Private Sub muda1_Click()
Form1.Visible = False
Form2.Visible = True
End Sub

Private Sub Option1_Click()
Option2.value = False
cmd1.Visible = True
cmd2.Visible = False
```

```
cmd3.Visible = False
Timer2.Enabled = False
End Sub

Private Sub Option2_Click()
Option1.value = False
cmd1.Visible = False
cmd2.Visible = True
cmd3.Visible = True
End Sub

Private Sub ServerCMD_Click()
Me.Caption = "Conectar como Servidor" 'renomeia
caption

IPtxt.Enabled = True 'botões
necessários/desnecessários para conectar
nicknameTXT.Enabled = True
ServerCMD.Enabled = False
ClientCMD.Enabled = False
IPtxt.Enabled = False
ConnectCMD.Enabled = True
CS = 1   'a aplicação passa a ser servidor

End Sub


Private Sub Timer_estados_Timer()
Dim i As Integer
i = 1

frase = "" 'zerando mensagem a ser enviada com os
estados

While (i < 57)
frase = frase & estados(i)
i = i + 1
Wend

txtestados.Text = frase 'TESTE
ServerFRM.msg.Text = frase 'após montar o string,
envia para o cliente
End Sub

Private Sub Timer_fim_Timer()
Timer1.Enabled = False
Timer2.Enabled = False
Timer3.Enabled = False
Timer_estados.Enabled = False
Form2.Timer3.Enabled = False
ClientFRM.t_c.Enabled = False
ServerFRM.t_s.Enabled = False
End 'finaliza totalmente o programa
End Sub

Private Sub Timer_load_Timer()
   'Este timer é para configurações iniciais
   contador_load = contador_load + 1
   cont.Text = contador_load 'TESTE

   If (contador_load = 1) Then
   'Zerando endereços do CLP de 0 a 67 (menos o
endereco 1)
   n = 0
   str = bin2.Text
   value_byteM(0) = binstr2value(str)
   While (n < 68)
     no = n
     res = m_field_write(no, AMOUNT, value_byteM(0))
     n = n + 1
     If (n = 1) Then
       n = n + 1 'pula endereco 1
     End If
   Wend
   End If 'contador_load = 1

   If (contador_load = 2) Then
   'passo 1 (conf inicial transporte e montagem)
   flag = 1
   bin3.Text = "11000000"   'Ativa 1.0 e 1.1 ao mesmo
tempo
   str = bin3.Text
   value_byteM(0) = binstr2value(str)
   no = flag
   res = m_field_write(no, AMOUNT, value_byteM(0))
   flag = 2
   End If

   If (contador_load = 3) Then
   'conf inicial 2
   str = bin1.Text 'Ativa 2.0
   value_byteM(0) = binstr2value(str)
   no = flag
   res = m_field_write(no, AMOUNT, value_byteM(0))
```

```
    flag = 1
    End If
    If (contador_load = 20) Then
    'ativa M1.2: conf motores para refference point e gira
a garra para 0 graus
    bin3.Text = "11100000"
    str = bin3.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    End If

    If (contador_load = 24) Then
    'ativa M1.3: start para refference point
    bin3.Text = "11110000"
    str = bin3.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    End If

    If (contador_load = 48) Then
    'ativa M1.4: conf motores para modo Automatic
    bin3.Text = "11111000"
    str = bin3.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    End If

    If (contador_load = 50) Then
    'ativa M1.5: start para colocar garra na posicao inicial
da montagem
    bin3.Text = "11111100"
    str = bin3.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    End If

    If (contador_load = 58) Then
    'ativa M1.6: escolhe programa 1 (responsavel pelos
passos da montagem)
    bin3.Text = "11111110"
    str = bin3.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    End If

    If (contador_load = 59) Then
    'ativa M1.7: desabilita program selection
    bin3.Text = "11111111"
    str = bin3.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    flag = 4 'flag 3.0 e 3.1 são usados para escolha da
cor da peça
    End If

    If (contador_load = 60) Then
    'acknowledge all
    Timer_load.Enabled = False
    str = bin1.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    End If

End Sub

Private Sub Timer_trava_Timer()
'modifica variável trava
If (Shape3.FillColor = vbBlack And flag = 6 And
pega_cor.Text <> "Sem Peça") Then
    trava = 1
Else: trava = 0
End If
End Sub

Private Sub Timer1_Timer()
'Este timer preenche de preto os carros que tem sensor
de posição ativos
If (s3.FillColor = vbGreen) Then
    Shape1.FillColor = vbBlack
    estados(44) = 1
Else
    Shape1.FillColor = vbWhite
    estados(44) = 0
End If

If (s9.FillColor = vbGreen) Then
    Shape2.FillColor = vbBlack
```

```
    estados(45) = 1
Else
    Shape2.FillColor = vbWhite
    estados(45) = 0
End If
If (s15.FillColor = vbGreen) Then
    Shape3.FillColor = vbBlack
    estados(46) = 1
Else
    Shape3.FillColor = vbWhite
    estados(46) = 0
End If
If (s21.FillColor = vbGreen) Then
    Shape4.FillColor = vbBlack
    estados(47) = 1
Else
    Shape4.FillColor = vbWhite
    estados(47) = 0
End If
End Sub

Private Sub Timer2_Timer()
    Call PROX
End Sub

Private Sub Timer3_Timer()
  Call READ0
  Call READ1
  Call READ2
  Call READ3
  Call READ4
  Call READ5
End Sub

Private Sub READ0()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

BLOCKNO = 1
no = 0

res = d_field_read(BLOCKNO, no, AMOUNT,
value_byte0(0))

If (res = 0) Then
    Call ShowValue0
End If

If (res <> 0) Then
  txt0.Text = "Reading error" 'TESTE
End If

End Sub

Private Sub ShowValue0()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

For i = 0 To (AMOUNT - 1)
   s = ""

        j = 1
        s1 = ""
        While (j < 256)
          k = j And value_byte0(i)

          'condicao para sensores:
          If (k <> 0 And j = 2) Then
            Form1.s12.FillColor = vbGreen
            estados(24) = 1
          End If
          If (k = 0 And j = 2) Then
            Form1.s12.FillColor = vbRed
            estados(24) = 0
          End If
          If (k <> 0 And j = 1) Then
            Form1.s9.FillColor = vbGreen
            estados(23) = 1
          End If
          If (k = 0 And j = 1) Then
            Form1.s9.FillColor = vbRed
            estados(23) = 0
          End If
          'fim da condicao para sensores

          'condicao sensores mesa
          If (k <> 0 And j = 4) Then
            Form1.s11.FillColor = vbGreen
            estados(22) = 1
          End If
          If (k = 0 And j = 4) Then
            Form1.s11.FillColor = vbRed
```

```
            estados(22) = 0
          End If
          If (k <> 0 And j = 8) Then
            Form1.s10.FillColor = vbGreen
            estados(21) = 1
          End If
          If (k = 0 And j = 8) Then
            Form1.s10.FillColor = vbRed
            estados(21) = 0
          End If
          'fim condicao mesa

          If (k <> 0) Then
            s1 = s1 + "1"
            Else
            s1 = s1 + "0"

          End If 'if do k
          j = j * 2
        Wend

  s = s + s1

  txt0.Text = (s) 'TESTE
Next i

End Sub

Private Sub READ1()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

BLOCKNO = 1
no = 1

res = d_field_read(BLOCKNO, no, AMOUNT,
value_byte1(0))

If (res = 0) Then
  Call ShowValue1
End If

If (res <> 0) Then
  txt1.Text = "Reading error" 'TESTE
End If
End Sub

Private Sub ShowValue1()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

For i = 0 To (AMOUNT - 1)
  s = ""

      j = 1
      s1 = ""
      While (j < 256)
        k = j And value_byte1(i)

        'condicao para sensores:
        If (k <> 0 And j = 32) Then
          Form1.s8.FillColor = vbGreen
          estados(20) = 1
        End If
        If (k = 0 And j = 32) Then
          Form1.s8.FillColor = vbRed
          estados(20) = 0
        End If
        If (k <> 0 And j = 16) Then
          Form1.s7.FillColor = vbGreen
          estados(19) = 1
        End If
        If (k = 0 And j = 16) Then
          Form1.s7.FillColor = vbRed
          estados(19) = 0
        End If
        'fim da condicao para sensores

        'condicao para pistoes
        If (k <> 0 And j = 1) Then
          Form1.p1.FillColor = vbWhite
          estados(1) = 1
        End If
        If (k = 0 And j = 1) Then
          Form1.p1.FillColor = vbCyan
          estados(1) = 0
        End If
        If (k <> 0 And j = 2) Then
          Form1.p2.FillColor = vbWhite
          estados(2) = 1
        End If
```

```
    If (k = 0 And j = 2) Then
       Form1.p2.FillColor = vbCyan
       estados(2) = 0
    End If
    If (k <> 0 And j = 4) Then
       Form1.p3.FillColor = vbWhite
       estados(3) = 1
    End If
    If (k = 0 And j = 4) Then
       Form1.p3.FillColor = vbCyan
       estados(3) = 0
    End If
    If (k <> 0 And j = 8) Then
       Form1.p4.FillColor = vbWhite
       estados(4) = 1
    End If
    If (k = 0 And j = 8) Then
       Form1.p4.FillColor = vbCyan
       estados(4) = 0
    End If
    If (k <> 0 And j = 64) Then
       Form1.p5.FillColor = vbWhite
       estados(5) = 1
    End If
    If (k = 0 And j = 64) Then
       Form1.p5.FillColor = vbCyan
       estados(5) = 0
    End If
    If (k <> 0 And j = 128) Then
       Form1.p6.FillColor = vbWhite
       estados(6) = 1
    End If
    If (k = 0 And j = 128) Then
       Form1.p6.FillColor = vbCyan
       estados(6) = 0
    End If
    'fim condicao pistoes

    If (k <> 0) Then
       s1 = s1 + "1"
       Else
       s1 = s1 + "0"

    End If 'if do k
    j = j * 2
Wend
    s = s + s1

    txt1.Text = (s) 'TESTE
Next i

End Sub

Private Sub READ2()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

BLOCKNO = 1
no = 2

res = d_field_read(BLOCKNO, no, AMOUNT,
value_byte2(0))

If (res = 0) Then
   Call ShowValue2
End If

If (res <> 0) Then
  txt2.Text = "Reading error" 'TESTE
End If

End Sub

Private Sub ShowValue2()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

For i = 0 To (AMOUNT - 1)
   s = ""

        j = 1
        s1 = ""
        While (j < 256)
           k = j And value_byte2(i)

           'condicao para sensores:
           If (k <> 0 And j = 1) Then
              Form1.s13.FillColor = vbGreen
              estados(25) = 1
           End If
           If (k = 0 And j = 1) Then
```

```
    Form1.s13.FillColor = vbRed
    estados(25) = 0
End If
If (k <> 0 And j = 2) Then
    Form1.s14.FillColor = vbGreen
    estados(26) = 1
End If
If (k = 0 And j = 2) Then
    Form1.s14.FillColor = vbRed
    estados(26) = 0
End If
 If (k <> 0 And j = 16) Then
    Form1.s15.FillColor = vbGreen
    estados(29) = 1
End If
If (k = 0 And j = 16) Then
    Form1.s15.FillColor = vbRed
    estados(29) = 0
End If
If (k <> 0 And j = 32) Then
    Form1.s18.FillColor = vbGreen
    estados(30) = 1
End If
If (k = 0 And j = 32) Then
    Form1.s18.FillColor = vbRed
    estados(30) = 0
End If
'fim da condicao para sensores

'condicao sensores mesa
If (k <> 0 And j = 64) Then
    Form1.s17.FillColor = vbGreen
    estados(28) = 1
End If
If (k = 0 And j = 64) Then
    Form1.s17.FillColor = vbRed
    estados(28) = 0
End If
If (k <> 0 And j = 128) Then
    Form1.s16.FillColor = vbGreen
    estados(27) = 1
End If
If (k = 0 And j = 128) Then
    Form1.s16.FillColor = vbRed
    estados(27) = 0
End If
'fim condicao mesa

                'indicadores de movimento dos motores
                If (k <> 0 And j = 4) Then
                    Form2.c1.Visible = True
                    Form2.c0.Visible = False
                    estados(48) = 1
                End If
                If (k = 0 And j = 4) Then
                    Form2.c1.Visible = False
                    Form2.c0.Visible = True
                    estados(48) = 0
                End If
                If (k <> 0 And j = 8) Then
                    Form2.d1.Visible = True
                    Form2.do.Visible = False
                    estados(49) = 1
                End If
                If (k = 0 And j = 8) Then
                    Form2.d1.Visible = False
                    Form2.do.Visible = True
                    estados(49) = 0
                End If
                'fim indicadores de movimento dos motores

                If (k <> 0) Then
                    s1 = s1 + "1"
                    Else
                    s1 = s1 + "0"

                End If 'if do k
                j = j * 2
            Wend

    s = s + s1

    txt2.Text = (s) 'TESTE
Next i

End Sub

Private Sub READ3()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

BLOCKNO = 1
no = 3
```

```
res = d_field_read(BLOCKNO, no, AMOUNT,
value_byte3(0))

If (res = 0) Then
   Call ShowValue3
End If

If (res <> 0) Then
  txt3.Text = "Reading error" 'TESTE
End If

End Sub

Private Sub ShowValue3()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

For i = 0 To (AMOUNT - 1)
   s = ""

        j = 1
        s1 = ""
        While (j < 256)
           k = j And value_byte3(i)

           'condicao para sensores:
           If (k <> 0 And j = 1) Then
              Form1.s21.FillColor = vbGreen
              estados(35) = 1
           End If
           If (k = 0 And j = 1) Then
              Form1.s21.FillColor = vbRed
              estados(35) = 0
           End If
           If (k <> 0 And j = 2) Then
              Form1.s24.FillColor = vbGreen
              estados(36) = 1
           End If
           If (k = 0 And j = 2) Then
              Form1.s24.FillColor = vbRed
              estados(36) = 0
           End If
           'fim da condicao para sensores

           'condicao sensores mesa
           If (k <> 0 And j = 4) Then
              Form1.s22.FillColor = vbGreen
              estados(33) = 1
           End If
           If (k = 0 And j = 4) Then
              Form1.s22.FillColor = vbRed
              estados(33) = 0
           End If
           If (k <> 0 And j = 8) Then
              Form1.s23.FillColor = vbGreen
              estados(34) = 1
           End If
           If (k = 0 And j = 8) Then
              Form1.s23.FillColor = vbRed
              estados(34) = 0
           End If
           'fim condicao mesa

           'condicao timer
           If (k <> 0 And j = 128) Then
              Form1.wait.Visible = True
              estados(52) = 1
           End If
           If (k = 0 And j = 128) Then
              Form1.wait.Visible = False
              estados(52) = 0
           End If
           'fim condicao timer



           'condicao pinos
           If (k <> 0 And j = 64) Then
              Form2.pino.FillColor = vbBlack
              estados(54) = 1
           End If
           If (k <> 0 And j = 32) Then
              Form2.pino.FillColor = &HC0C0C0
              estados(54) = 0
           End If
           'fim condicao pinos

           If (k <> 0) Then
              s1 = s1 + "1"
              Else
              s1 = s1 + "0"

           End If 'if do k
           j = j * 2
```

```
        Wend

   s = s + s1

   txt3.Text = (s) 'TESTE
Next i

End Sub

Private Sub READ4()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

BLOCKNO = 1
no = 4

res = d_field_read(BLOCKNO, no, AMOUNT,
value_byte4(0))

If (res = 0) Then
   Call ShowValue4
End If

If (res <> 0) Then
  txt4.Text = "Reading error" 'TESTE
End If

End Sub

Private Sub ShowValue4()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

For i = 0 To (AMOUNT - 1)
   s = ""

        j = 1
        s1 = ""
        While (j < 256)
           k = j And value_byte4(i)

           'condicao para sensores:
           If (k <> 0 And j = 16) Then
              Form1.s19.FillColor = vbGreen
              estados(31) = 1
End If
If (k = 0 And j = 16) Then
   Form1.s19.FillColor = vbRed
   estados(31) = 0
End If

If (k <> 0 And j = 32) Then
   Form1.s20.FillColor = vbGreen
   estados(32) = 1
End If
If (k = 0 And j = 32) Then
   Form1.s20.FillColor = vbRed
   estados(32) = 0
End If
'fim da condicao para sensores

'condicao para pistoes
If (k <> 0 And j = 1) Then
   Form1.p7.FillColor = vbWhite
   estados(7) = 1
End If
If (k = 0 And j = 1) Then
   Form1.p7.FillColor = vbCyan
   estados(7) = 0
End If
If (k <> 0 And j = 2) Then
   Form1.p8.FillColor = vbWhite
   estados(8) = 1
End If
If (k = 0 And j = 2) Then
   Form1.p8.FillColor = vbCyan
   estados(8) = 0
End If
If (k <> 0 And j = 4) Then
   Form1.p9.FillColor = vbWhite
   estados(9) = 1
End If
If (k = 0 And j = 4) Then
   Form1.p9.FillColor = vbCyan
   estados(9) = 0
End If
If (k <> 0 And j = 8) Then
   Form1.p10.FillColor = vbWhite
   estados(10) = 1
End If
If (k = 0 And j = 8) Then
   Form1.p10.FillColor = vbCyan
```

```
            estados(10) = 0
          End If
          If (k <> 0 And j = 64) Then
            Form1.p11.FillColor = vbWhite
            estados(11) = 1
          End If
          If (k = 0 And j = 64) Then
            Form1.p11.FillColor = vbCyan
            estados(11) = 0
          End If
          If (k <> 0 And j = 128) Then
            Form1.p12.FillColor = vbWhite
            estados(12) = 1
          End If
          If (k = 0 And j = 128) Then
            Form1.p12.FillColor = vbCyan
            estados(12) = 0
          End If
          'fim condicao pistoes

          If (k <> 0) Then
            s1 = s1 + "1"
            Else
            s1 = s1 + "0"

          End If 'if do k
          j = j * 2
        Wend

   s = s + s1

   txt4.Text = (s) 'TESTE
Next i

End Sub

Private Sub READ5()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

BLOCKNO = 1
no = 5

res = d_field_read(BLOCKNO, no, AMOUNT,
value_byte5(0))

If (res = 0) Then
   Call ShowValue5
End If

If (res <> 0) Then
  txt5.Text = "Reading error" 'TESTE
End If

End Sub

Private Sub ShowValue5()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

For i = 0 To (AMOUNT - 1)
   s = ""

        j = 1
        s1 = ""
        While (j < 256)
          k = j And value_byte5(i)

          'condicao para sensores:
          If (k <> 0 And j = 1) Then
            Form1.s1a.FillColor = vbGreen
            estados(13) = 1
          End If
          If (k = 0 And j = 1) Then
            Form1.s1a.FillColor = vbRed
            estados(13) = 0
          End If
          If (k <> 0 And j = 2) Then
            Form1.s2.FillColor = vbGreen
            estados(14) = 1
          End If
          If (k = 0 And j = 2) Then
            Form1.s2.FillColor = vbRed
            estados(14) = 0
          End If
          If (k <> 0 And j = 16) Then
            Form1.s3.FillColor = vbGreen
            estados(17) = 1
          End If
          If (k = 0 And j = 16) Then
            Form1.s3.FillColor = vbRed
            estados(17) = 0
```

```
End If
If (k <> 0 And j = 32) Then
   Form1.s6.FillColor = vbGreen
   estados(18) = 1
End If
If (k = 0 And j = 32) Then
   Form1.s6.FillColor = vbRed
   estados(18) = 0
End If
'fim da condicao para sensores

'condicao sensores mesa
If (k <> 0 And j = 64) Then
   Form1.s4.FillColor = vbGreen
   estados(15) = 1
End If
If (k = 0 And j = 64) Then
   Form1.s4.FillColor = vbRed
   estados(15) = 0
End If
If (k <> 0 And j = 128) Then
   Form1.s5.FillColor = vbGreen
   estados(16) = 1
End If
If (k = 0 And j = 128) Then
   Form1.s5.FillColor = vbRed
   estados(16) = 0
End If
'fim condicao mesa

'condicao indicadores movimento motores
If (k <> 0 And j = 4) Then
   Form2.b1.Visible = True
   Form2.b0.Visible = False
   estados(50) = 1
End If
If (k = 0 And j = 4) Then
   Form2.b1.Visible = False
   Form2.b0.Visible = True
   estados(50) = 0
End If
If (k <> 0 And j = 8) Then
   Form2.e1.Visible = True
   Form2.e0.Visible = False
   estados(51) = 1
End If
If (k = 0 And j = 8) Then
   Form2.e1.Visible = False
   Form2.e0.Visible = True
   estados(51) = 0
End If
'fim condicao movimento motores

If (k <> 0) Then
   s1 = s1 + "1"
   Else
   s1 = s1 + "0"

End If 'if do k
j = j * 2
Wend

  s = s + s1

  txt5.Text = (s) 'TESTE
Next i

End Sub
```

C5) Form 2:

Este form representa a interface da estação de montagem. Neste form são feitas as lógicas para o controle local e remoto e para a monitoração do sistema, alterando os estados dos sensores representados.

```
Private Sub cmd1_Click()
 If (CS = 0) Then
  Call PROX2
 End If 'CS = 0

 If (CS = 2) Then
  ClientFRM.msg.Text = "next2"
 End If
End Sub

Private Sub cmd2_Click()
If (CS = 0) Then
 Timer2.Enabled = True
 cmd2.Enabled = False
 Option1.Enabled = False 'Garante que usuário não
mude para manual sem desligar o automático
End If
```

```
If (CS = 2) Then
  ClientFRM.msg.Text = "start2"
  cmd2.Enabled = False
  Option1.Enabled = False 'Garante que usuário não
mude para manual sem desligar o automático
End If
End Sub

Private Sub cmd3_Click()
If (CS = 0) Then
  Timer2.Enabled = False
  cmd2.Enabled = True
  Option1.Enabled = True
End If

If (CS = 2) Then
  ClientFRM.msg.Text = "stop2"
  cmd2.Enabled = True
  Option1.Enabled = True
End If
End Sub

Private Sub PROX2()
  Dim m As Integer
  'estação de montagem só movimenta se:
  '1) Estação de transporte está no flag = 6
  If (flag = 6 And indicador = 0) Then
    '2) Há carro na posição C
    If (Form1.Shape3.FillColor = vbBlack) Then
      '3a) Há peça no carro
      If (Form1.pega_cor.Text <> "Sem Peça" Or flag2 >
8) Then
        flag2 = flag2 + 1
        If (flag2 = 68) Then
          indicador = 1
          m = 9
          str = Form1.bin2.Text
          value_byteM(0) = binstr2value(str)
          While (m < 68)
            no = m
            res = m_field_write(no, AMOUNT,
value_byteM(0))
            m = m + 1
          Wend
          flag2 = 8 'O programa fica preso nos passos 9 a
67
        End If 'flag2 = 68
        If (flag2 <> 8) Then
        txt2_timer2.Text = flag2 'TESTE
        str = Form1.bin1.Text
        value_byteM(0) = binstr2value(str)
        no = flag2
        res = m_field_write(no, AMOUNT,
value_byteM(0))
        End If
        If (flag2 = 9) Then
          Form1.pega_cor.Text = "Sem Peça"
          Form1.peca.FillColor = vbWhite
        End If

        'indicadores de posição da garra
        If (flag2 = 9 Or flag2 = 65) Then
          g1.Visible = True
          g2.Visible = False
          g3.Visible = False
          g4.Visible = False
          g5.Visible = False
          g6.Visible = False
          estados(55) = 1
        End If
        If (flag2 = 13 Or flag2 = 57) Then
          g1.Visible = False
          g2.Visible = True
          g3.Visible = False
          g4.Visible = False
          g5.Visible = False
          g6.Visible = False
          estados(55) = 2
        End If
        If (flag2 = 21 Or flag2 = 31 Or flag2 = 40 Or flag2 =
51) Then
          g1.Visible = False
          g2.Visible = False
          g3.Visible = True
          g4.Visible = False
          g5.Visible = False
          g6.Visible = False
          estados(55) = 3
        End If
        If (flag2 = 27) Then
          g1.Visible = False
          g2.Visible = False
          g3.Visible = False
```

```
        g4.Visible = True
        g5.Visible = False
        g6.Visible = False
        estados(55) = 4
      End If
      If (flag2 = 35) Then
        g1.Visible = False
        g2.Visible = False
        g3.Visible = False
        g4.Visible = False
        g5.Visible = True
        g6.Visible = False
        estados(55) = 5
      End If
      If (flag2 = 45) Then
        g1.Visible = False
        g2.Visible = False
        g3.Visible = False
        g4.Visible = False
        g5.Visible = False
        g6.Visible = True
        estados(55) = 6
      End If
      If (flag2 = 10 Or flag2 = 14 Or flag2 = 18 Or flag2 =
22 Or flag2 = 23 Or flag2 = 25 Or flag2 = 28 Or flag2 =
32 Or flag2 = 35 Or flag2 = 37 Or flag2 = 41 Or flag2 =
43 Or flag2 = 45 Or flag2 = 46 Or flag2 = 48 Or flag2 =
51 Or flag2 = 53 Or flag2 = 54 Or flag2 = 58 Or flag2 =
62 Or flag2 = 66) Then
        Timer2.Interval = 1500
      Else
        Timer2.Interval = 5000
      End If
    End If 'sem peca
  End If 'carro posição C
 End If 'flag = 6
End Sub

Function binstr2value(s As String) As Long
Dim v As Long
Dim w As Long
Dim i, j As Long

v = 0
w = 1

j = Len(s)
For i = 1 To j
  If (Mid$(s, i, 1) = "1") Then v = v Or w
  w = w * 2
Next i
binstr2value = v
End Function

Private Sub emergencia1_Click()
  If (CS = 0) Then
  no = 0
  str = Form1.bin1.Text
  value_byteM(0) = binstr2value(str)
  res = m_field_write(no, AMOUNT, value_byteM(0))
  Form1.Timer2.Enabled = False
  Form2.Timer2.Enabled = False
  emergencia1.BackColor = vbRed
  End If

  If (CS = 2) Then
   ClientFRM.msg.Text = "emergencia"
   emergencia1.BackColor = vbRed
  End If
End Sub

Private Sub Form_Load()
flag2 = 8
indicador = 0 'impede que a montagem seja feita duas
vezes para a mesma peca
End Sub

Private Sub muda2_Click()
Form2.Visible = False
Form1.Visible = True
End Sub

Private Sub Option1_Click()
Option2.value = False
cmd1.Visible = True
cmd2.Visible = False
cmd3.Visible = False
Timer2.Enabled = False
End Sub

Private Sub Option2_Click()
Option1.value = False
cmd1.Visible = False
cmd2.Visible = True
```

```
cmd3.Visible = True
End Sub

Private Sub Timer2_Timer()
  Call PROX2
End Sub

Private Sub Timer3_Timer()
  Call READ1
  Call READ2
End Sub

Private Sub READ1()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

no = 20

res = e_field_read(no, AMOUNT, value_byte(0))

If (res = 0) Then
   Call ShowValue1
End If

If (res <> 0) Then
  txtread3.Text = "Reading error" 'TESTE
End If

End Sub

Private Sub ShowValue1()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

For i = 0 To (AMOUNT - 1)
   s = ""

        j = 1
        s1 = ""
        While (j < 256)
           k = j And value_byte(i)

           'condicao para sensores:
           If (k <> 0 And j = 64) Then
              b15.FillColor = vbGreen
              estados(42) = 1
           End If
           If (k = 0 And j = 64) Then
              b15.FillColor = vbRed
              estados(42) = 0
           End If
           If (k <> 0 And j = 32) Then
              b14.FillColor = vbGreen
              estados(41) = 1
           End If
           If (k = 0 And j = 32) Then
              b14.FillColor = vbRed
              estados(41) = 0
           End If
           If (k <> 0 And j = 8) Then
              b12.FillColor = vbGreen
              estados(43) = 1
           End If
           If (k = 0 And j = 8) Then
              b12.FillColor = vbRed
              estados(43) = 0
           End If
           'fim da condicao para sensores

           If (k <> 0) Then
              s1 = s1 + "1"
              Else
              s1 = s1 + "0"

           End If 'if do k
           j = j * 2
        Wend

   s = s + s1

   txtread3.Text = (s) 'TESTE

Next i

End Sub

Private Sub READ2()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

no = 16
```

```
res = e_field_read(no, AMOUNT, value_byteA(0))

If (res = 0) Then
   Call ShowValue2
End If

If (res <> 0) Then
  txtread4.Text = "Reading error" 'TESTE
End If

End Sub

Private Sub ShowValue2()
Dim i, j, k As Long
Dim s As String
Dim s1 As String

For i = 0 To (AMOUNT - 1)
   s = ""

        j = 1
        s1 = ""
        While (j < 256)
           k = j And value_byteA(i)

           'condicao para sensores:
           If (k <> 0 And j = 128) Then
              b27.FillColor = vbGreen
              estados(37) = 1
           End If
           If (k = 0 And j = 128) Then
              b27.FillColor = vbRed
              estados(37) = 0
           End If
           If (k <> 0 And j = 64) Then
              b26.FillColor = vbGreen
              estados(38) = 1
           End If
           If (k = 0 And j = 64) Then
              b26.FillColor = vbRed
              estados(38) = 0
           End If
           If (k <> 0 And j = 32) Then
              b25.FillColor = vbGreen
              estados(39) = 1
           End If
           If (k = 0 And j = 32) Then
              b25.FillColor = vbRed
              estados(39) = 0
           End If
           If (k <> 0 And j = 16) Then
              b24.FillColor = vbGreen
              estados(40) = 1
           End If
           If (k = 0 And j = 16) Then
              b24.FillColor = vbRed
              estados(40) = 0
           End If
           'fim da condicao para sensores

           'condicao sensores garra
           If (k <> 0 And j = 1) Then
              garraf.Visible = True
              garraa.Visible = False
              estados(53) = 0
           End If
           If (k = 0 And j = 1) Then
              garraf.Visible = False
              garraa.Visible = True
              estados(53) = 1
           End If
           If (k <> 0 And j = 8) Then
              garra270.Visible = True
              garra0.Visible = False
              estados(56) = 1
           End If
           If (k = 0 And j = 8) Then
              garra270.Visible = False
              garra0.Visible = True
              estados(56) = 0
           End If
           'fim condicao sensores garra
           If (k <> 0) Then
              s1 = s1 + "1"
              Else
              s1 = s1 + "0"

           End If 'if do k
           j = j * 2
        Wend

s = s + s1
```

```
    txtread4.Text = (s) 'TESTE
Next i

End Sub
```

C6) ok:

Este form apresenta a mensagem de confirmação da conexão entre computador local e CP.

```
Private Sub cmdok_Click()
ok.Visible = False
End Sub
```

C7) ServerFRM:

Este form é responsável pelas operações do computador local. As principais tarefas são a troca de dados via *chat* entre operadores, envio do vetor de estados dos sensores e atuadoes e envio de comandos à interface remota.

```
Public Sub s_server()
Winsock.SendData "M" & msg.Text 'envia dados ao cliente
msg.Text = ""
End Sub

Private Sub t_s_Timer()
Call s_server
End Sub

'=======================================================================
'======================Exibe informações se conectado=======================
Private Sub Winsock_Close()
    MainTXT.SelText = ""¨º*.ª.*º¨" Desconectado do Cliente ¨º*.ª.*º¨" & vbCrLf 'exibe texto se for desconectado do cliente"
    senddataCMD.Enabled = False 'não permite enviar mais dados ao cliente caso desconectado
End Sub
'=======================================================================
'======================Permite o cliente conectar==========================

Private Sub winsock_ConnectionRequest(ByVal requestID As Long)

If Winsock.State <> sckClosed Then Winsock.Close 'fecha conexão se existe uma aberta
Winsock.Accept requestID 'permite nova conexão

t_s.Enabled = True 'inicia loop para envio de msg

Me.Show 'exibe form servidor
Winsock.SendData "C" & nickSERVER.Caption  'envia dado ao cliente para carregar form
ServerFRM.Caption = "Chat   [Bem vindo, " & nickSERVER.Caption & "!]" 'renomeia o form
MainTXT.SelText = ""¨º*.ª.*º¨ Conectado ao Cliente ¨º*.ª.*º¨" & vbCrLf 'mostra que a conexão foi estabelecida

End Sub
'=======================================================================
'======================Recebe dados do cliente=============================
Private Sub winsock_DataArrival(ByVal bytesTotal As Long)
Dim strData, strData2 As String 'onde o dado recebido é armazenado
Call Winsock.GetData(strData, vbString) 'pega dados recebidos

strData2 = Left(strData, 1) 'salva primeira letra da mensagem
strData = Mid(strData, 2) 'salva o restante da mensagem

If strData2 = "T" Then MainTXT.SelText = nickCLIENT.Caption & ":     " & strData & vbCrLf 'adiciona o dado ao textbox
If strData2 = "N" Then nickCLIENT.Caption = strData 'carrega o nome do cliente

'recebendo comandos do cliente
If strData2 = "M" Then
  If strData = "load" Then
    plcadr(0).adr = 2
    plcadr(0).SEGMENTID = 0
    plcadr(0).RACKNO = 0
```

```
    plcadr(0).SLOTNO = 2
    plcadr(1).adr = 0
    VerbIdx = 0
    res = load_tool(1, "S7ONLINE", plcadr(0))
    If (res = 0) Then
    contador_load = 0
    Form1.Timer_load.Enabled = True
    msg.Text = "ok"
    'como servidor, ativo leitura dos estados:
    Form1.Timer_estados.Enabled = True
    End If 'res = 0
    If (res <> 0) Then msg.Text = "cerror" 'erro de
conexão
  End If 'if = "load"

  If strData = "unload" Then
    res = unload_tool()
    If (res = 0) Then
       Form1.Timer_estados.Enabled = False
       msg.Text = "unload_ok"
    End If
  End If 'if = "unload"

  If strData = "start" Then
    Form1.Timer2.Enabled = True
  End If

  If strData = "next" Then
     Call PROX
  End If

  If strData = "stop" Then
    Form1.Timer2.Enabled = False
  End If

  If strData = "start2" Then
    Form2.Timer2.Enabled = True
  End If

  If strData = "next2" Then
     Call PROX2
  End If

  If strData = "stop2" Then
    Form2.Timer2.Enabled = False
  End If

If strData = "finalizar" Then
  Form1.Timer1.Enabled = False
  Form1.Timer2.Enabled = False
  Form1.Timer3.Enabled = False
  Form1.Timer_estados.Enabled = False
  Form2.Timer3.Enabled = False
  ClientFRM.t_c.Enabled = False
  ServerFRM.t_s.Enabled = False
  End 'finaliza totalmente o programa
End If

If strData = "preto" Then
  Form1.pega_cor.Text = "Preto"
  'altera cor do pino: 1 prata, 0 preto
  Form1.bin3.Text = "01000000"
  str = Form1.bin3.Text
  value_byteM(0) = binstr2value(str)
  no = 3
  res = m_field_write(no, AMOUNT, value_byteM(0))
End If

If strData = "prata" Then
  Form1.pega_cor.Text = "Prata"
  'altera cor do pino: 0 prata, 1 preto
  Form1.bin3.Text = "10000000"
  str = Form1.bin3.Text
  value_byteM(0) = binstr2value(str)
  no = 3
  res = m_field_write(no, AMOUNT, value_byteM(0))
End If

If strData = "rosa" Then
  Form1.pega_cor.Text = "Rosa"
  'altera cor do pino: 0 prata, 1 preto
  Form1.bin3.Text = "10000000"
  str = Form1.bin3.Text
  value_byteM(0) = binstr2value(str)
  no = 3
  res = m_field_write(no, AMOUNT, value_byteM(0))
End If

If strData = "sem_peca" Then
  Form1.pega_cor.Text = "Sem Peça"
End If

If strData = "emergencia" Then
  no = 0
```

```
    str = Form1.bin1.Text
    value_byteM(0) = binstr2value(str)
    res = m_field_write(no, AMOUNT, value_byteM(0))
    Form1.Timer2.Enabled = False
    Form2.Timer2.Enabled = False
  End If
End If  'if = "M"
End Sub

'=======================================
==================================
'=======================Envia dados ao
cliente============================
Private Sub senddataCMD_Click()
MainTXT.SelText = nickSERVER.Caption & ":     " &
dataTXT.Text & vbCrLf 'insere texto digitado no textbox
Winsock.SendData "T" & dataTXT.Text 'envia dados ao
cliente
dataTXT.Text = "" 'apaga mensagem digitada
End Sub
'=======================================
==================================

Private Sub PROX()
  Dim m As Integer
  'estação de transporte só movimenta se acabou
montagem
  If (flag2 = 8 And trava = 0) Then
    flag = flag + 1
    If (flag = 9) Then
      m = 5
      str = Form1.bin2.Text
      value_byteM(0) = binstr2value(str)
      While (m < 9)
        no = m
        res = m_field_write(no, AMOUNT,
value_byteM(0))
        m = m + 1
      Wend
      flag = 5 'O programa fica preso nos passos 5 a 8
    End If 'flag = 9
    Form1.txt_timer2.Text = flag 'TESTE
    If (flag = 6) Then
      Form1.Timer1.Enabled = False
      indicador = 0
    End If
    str = Form1.bin1.Text
    value_byteM(0) = binstr2value(str)
    no = flag
    res = m_field_write(no, AMOUNT, value_byteM(0))
    If (flag = 7) Then Form1.Timer1.Enabled = True
  End If 'flag2 = 8
End Sub

Function binstr2value(s As String) As Long
Dim v As Long
Dim w As Long
Dim i, j As Long
v = 0
w = 1
j = Len(s)
For i = 1 To j
    If (Mid$(s, i, 1) = "1") Then v = v Or w
    w = w * 2
Next i
binstr2value = v
End Function

Private Sub PROX2()
  Dim m As Integer
  'estação de montagem só movimenta se:
  '1) Estação de transporte está no flag = 6
  If (flag = 6 And indicador = 0) Then
    '2) Há carro na posição C
    If (Form1.Shape3.FillColor = vbBlack) Then
      '3a) Há peça no carro
      If (Form1.pega_cor.Text <> "Sem Peça" Or flag2 >
8) Then
        flag2 = flag2 + 1
        If (flag2 = 68) Then
          indicador = 1
          m = 9
          str = Form1.bin2.Text
          value_byteM(0) = binstr2value(str)
          While (m < 68)
            no = m
            res = m_field_write(no, AMOUNT,
value_byteM(0))
            m = m + 1
          Wend
          flag2 = 8 'O programa fica preso nos passos 9 a
67
        End If 'flag2 = 68
        If (flag2 <> 8) Then
```

```
        Form2.txt2_timer2.Text = flag2 'TESTE
        str = Form1.bin1.Text
        value_byteM(0) = binstr2value(str)
        no = flag2
        res = m_field_write(no, AMOUNT,
value_byteM(0))
        End If
        If (flag2 = 9) Then
           Form1.pega_cor.Text = "Sem Peça"
           Form1.peca.FillColor = vbWhite
        End If
        'indicadores de posição da garra
        If (flag2 = 9 Or flag2 = 65) Then
          Form2.g1.Visible = True
          Form2.g2.Visible = False
          Form2.g3.Visible = False
          Form2.g4.Visible = False
          Form2.g5.Visible = False
          Form2.g6.Visible = False
          estados(55) = 1
        End If
        If (flag2 = 13 Or flag2 = 57) Then
          Form2.g1.Visible = False
          Form2.g2.Visible = True
          Form2.g3.Visible = False
          Form2.g4.Visible = False
          Form2.g5.Visible = False
          Form2.g6.Visible = False
          estados(55) = 2
        End If
        If (flag2 = 21 Or flag2 = 31 Or flag2 = 40 Or flag2 =
51) Then
          Form2.g1.Visible = False
          Form2.g2.Visible = False
          Form2.g3.Visible = True
          Form2.g4.Visible = False
          Form2.g5.Visible = False
          Form2.g6.Visible = False
          estados(55) = 3
        End If
        If (flag2 = 27) Then
          Form2.g1.Visible = False
          Form2.g2.Visible = False
          Form2.g3.Visible = False
          Form2.g4.Visible = True
          Form2.g5.Visible = False
          Form2.g6.Visible = False
          estados(55) = 4
        End If
        If (flag2 = 35) Then
          Form2.g1.Visible = False
          Form2.g2.Visible = False
          Form2.g3.Visible = False
          Form2.g4.Visible = False
          Form2.g5.Visible = True
          Form2.g6.Visible = False
          estados(55) = 5
        End If
        If (flag2 = 45) Then
          Form2.g1.Visible = False
          Form2.g2.Visible = False
          Form2.g3.Visible = False
          Form2.g4.Visible = False
          Form2.g5.Visible = False
          Form2.g6.Visible = True
          estados(55) = 6
        End If
        If (flag2 = 10 Or flag2 = 14 Or flag2 = 18 Or flag2 =
22 Or flag2 = 23 Or flag2 = 25 Or flag2 = 28 Or flag2 =
32 Or flag2 = 35 Or flag2 = 37 Or flag2 = 41 Or flag2 =
43 Or flag2 = 45 Or flag2 = 46 Or flag2 = 48 Or flag2 =
51 Or flag2 = 53 Or flag2 = 54 Or flag2 = 58 Or flag2 =
62 Or flag2 = 66) Then
          Form2.Timer2.Interval = 1500
        Else
          Form2.Timer2.Interval = 5000
        End If
      End If 'sem peca
    End If 'carro posição C
  End If 'flag = 6
End Sub
```

C8) unload:

Este form apresenta a confirmação da desconexão entre computador local e CP.

```
Private Sub cmdunload_Click()
unload.Visible = False
End Sub
```