BRUNO DE LIMA PINHEIRO

Implementação de um ambiente para a simulação distribuída de sistemas produtivos

São Paulo
2006

BRUNO DE LIMA PINHEIRO

nota final 9,4
(nove e quatro)

Implementação de um ambiente para a simulação distribuída de sistemas produtivos

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

Curso de Graduação: Engenharia Mecatrônica

Orientador: Prof. Dr. Paulo Eigi Miyagi

São Paulo
2006

FICHA CATALOGRÁFICA

Pinheiro, Bruno de Lima
Implementação de um ambiente para a simulação distribuída
de sistemas produtivos / B. de L. Pinheiro. – São Paulo, 2006.
90 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Administração da produção 2.Simulação distribuída 3.Rede
de Petri I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

A Deus e aos meus pais,

José Antônio e Anamaria.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Paulo Eigi Miyagi, pela sua constante orientação e incentivo para o desenvolvimento deste trabalho.

Aos colegas de sala Ana, Carlos, César, Alexandre e Rafael pela ajuda e companhia.

À Ruth, pelo seu amor e carinho.

À Escola Politecnica da USP, em especial ao Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, que institucionalmente viabilizaram este trabalho.

Tudo que podia ser inventado já o foi.
(Charles Duell, diretor Depto. Patentes EUA, 1899)

RESUMO

Os sistemas produtivos nos países industrializados têm evoluído para uma nova maneira de operar, impulsionada pelo desenvolvimento de tecnologias de telecomunicação, mecatrônica e mobilidade. Esse novo sistema produtivo tem, entre suas principais características, a descentralização das tarefas e uma maior especialização de cada tipo de processo; tudo isso aliado a uma maior automatização das operações gera sistemas que envolvem interações relativamente complexas entre partes que podem estar distribuídas numa mesma planta ou até mesmo dispersas geograficamente.

Para análise das soluções de projeto desses sistemas, uma das abordagens que tem sido considerada é a simulação distribuída, a qual trata da execução de modelos instalados em computadores dispersos, conectados através de uma rede de comunicação. Esta abordagem envolve o desenvolvimento de um ambiente distribuído, baseado, por exemplo, em redes de Petri, para a descrição e análise dos processos. Este ambiente deve assim dar suporte à concepção de sistemas de telecomando e monitoramento remoto, permitindo que diferentes módulos do sistema sejam desenvolvidos por diferentes pessoas em diferentes localidades, e que o comportamento destes módulos possa ser analisado via simulação.

Portanto, o objetivo deste trabalho é o desenvolvimento e teste de algoritmos de gerenciamento de processos de simulação distribuída usando comunicação entre computadores via sockets e/ou middlewares (ex. CORBA).

Como estudo de caso, considerou-se o sistema flexível de montagem onde cada estação de trabalho pode ser entendida como uma instalação produtiva distinta.

Palavras chaves: telecomando,monitoração remota, simulação distribuída, rede de Petri.

ABSTRACT

The productive systems in the industrialized countries have evolved for a new way to operate, stimulated for the development of technologies of telecommunication, mechatronics and mobility. This new productive system has, among its main characteristics, the decentralization of the tasks and an increasing specialization of each type of process; besides this the automatization of the operations require control systems that involve relatively complex interactions between parts that can be distributed in one same plant or even though geographically dispersed.

For analysis of these systems, one of the approaches that has been considered is the distributed simulation, which deals with the execution of models installed in dispersed computers, connected through a communication net. This approach involves the development of a distributed environment, based for example, in Petri nets for the description and analysis of the processes. This environment must be a support to the conception of telecommand and remote monitoring systems, allowing that different modules of the system be developed by different people in different localities, and that the behavior of these modules be analyzed through simulation.

Therefore, the objective of this work is the development and test of algorithms for processes management of distributed simulation using the communication among computers using sockets and/or middlewares (ex. CORBA).

As a case study, the flexible assembly system where each station of work can be understood as a distinct productive installation was considered.

Key Words: telecommand, remote monitoring, distributed simulation, Petri net.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE PALAVRAS RESERVADAS

Redes de Petri (**Fonte**: Times New Roman 12)

- Transição
- Lugar
- Marca
- Arco habilitador
- Arco inibidor

Protocolo sockets (**Fonte**: Verdana 12)

- Sock
- servIP
- servPort
- connect
- closesocket
- send
- recv
- WSAStartup
- Listen

Funções específicas do programa implementado (Fonte: Tahoma 12)

- load_tool( )
- a_field_write( )
- e_field_read( )
- a_field_read( )

LISTA DE ABREVIATURAS E SIGLAS

API	Aplication program interface
CIM	Computer Integrated Manufacturing
CLP	Controle lógico programável
CORBA	Common object request broker architecture
FBD	Blocos de função
IDL	Interface definition language
LAD	Ladder
LAN	Local area network
OMA	Object architecture model
OMG	Object management group
OPN	Object Petri Net
ORB	Object request broker
Profibus DP	Periferia Descentralizada
RdP	Rede de Petri
SEDs	Sistemas a eventos discretos
SIT	Sistema Inteligente de Transporte
STL	Lista de instruções
WAN	Wide area network
XML	Extensible Markup Language
ZVEI	Associação Central da Indústria Elétrica Alemã

SUMÁRIO

1. INTRODUÇÃO

O quadro atual do setor produtivo é diferente do modelo Fordista da revolução industrial. Hoje devido a evolução tecnológica, o setor sofreu e sofre mudanças significativas na maneira de conceber e produzir bens e serviços. Os meios de produção deixam de ser em plantas únicas e lineares, para tornarem-se dispersas fisicamente e com inúmeros processos que são conduzidos de modo paralelo (JUNQUEIRA, 2006).

Os produtos não são mais feitos em fábricas verticalizadas de grande porte onde a matéria prima entra de um lado da linha de produção e o produto final sai pronto do outro lado. O sistema produtivo atual conta com várias partes (incluindo várias empresas), sendo cada uma delas especializada no seu setor, e os vários componentes dos produtos passam por elas de maneira concomitante, recebendo em cada etapa um trabalho específico (LEE;LAU,1999). Além disso, a competição no mercado não envolve apenas os preços, mas também o grau de inovação tecnológica dos produtos.

Isso tudo leva a meios de produção mais complexos, com maior grau de automatização, com tecnologias programáveis e flexíveis, mecanismos autônomos e sistemas distribuídos fisicamente (JUNQUEIRA, 2006).

A maneira de pensar da sociedade com relação aos sistemas produtivos também tem mudado, abandonando o foco de que a empresa é uma entidade isolada, passando agora a uma visão mais integrada e dinâmica, considerando a atuação de uma empresa dentro de uma rede corporativa internacional de fábricas (SHI; GREGORY, 1998).

Considerando, portanto, o aumento da complexidade destes novos sistemas produtivos, onde se fazem necessários o uso de novas técnicas de modelagem e análise, em particular a distribuída, o objetivo deste trabalho é a implementação de um ambiente de simulação distribuída de sistemas produtivos.

1.1. Organização do texto

No capítulo 2 são apresentados os fundamentos teóricos que embasam o trabalho, sendo feito um estudo das técnicas de modelagem e simulação e um estudo comparativo das linguagens CORBA e *sockets* para implementação do ambiente distribuído.

No capítulo 3 é apresentado o sistema produtivo MiniCIM instalado no PMR-EPUSP, bem como o estudo de seus componentes.

No capítulo 4 é proposto o ambiente de simulação distribuída de sistemas produtivos.

No capítulo 5 são apresentados os comentários finais, conclusões e propostas de trabalhos futuros.

2. TÉCNICAS DE MODELAGEM E SIMULAÇÃO

Para analisar, projetar e construir sistemas do ponto de vista de engenharia, onde diversas partes interagem para atingir um objetivo maior, são desenvolvidos modelos, sejam eles físicos, matemáticos, computacionais, etc. Isso porque, dependendo da dimensão e complexidade do sistema, fica inviável sua implementação real para posterior avaliação. Dessa maneira, o modelo é uma representação do sistema real, com suas principais partes e respectivas interações (EYKHOFF, 1974).

A modelagem pode ser definida como (CENTENO, 1996): a prática de se construir modelos para representar sistemas reais existentes, ou sistemas hipotéticos, e realizar experimentos com estes modelos para: (1) explicar o comportamento dos sistemas; (2) construir teorias ou hipóteses que consideram o comportamento observado; (3) aumentar o desempenho do sistema; (4) projetar novos sistemas com o desempenho desejado; e/ou (5) descrever comportamentos futuros ou o efeito produzido por mudanças no conjunto das entradas.

A simulação de experimentos por sua vez é uma das técnicas usadas para avaliar os sistemas. Nela o modelo recebe entradas similares às situações reais, para que seu desempenho seja estudado e diferentes soluções possam ser comparadas.

Com relação a outras técnicas de análise de modelos de sistemas, a simulação apresenta algumas vantagens. São elas (JUNQUEIRA, 2006):

- a capacidade de analisar modelos de complexidade arbitrária, ou seja, é uma técnica relativamente flexível, pois os modelos não precisam de muitas simplificações como nos casos analíticos;
- o custo dos dados coletados através de simulação são relativamente menores do que os obtidos através do sistema real;
- a análise por simulação pode ser desempenhada com o mínimo risco para os envolvidos na aquisição e operação do sistema, pois é possível realizar mudanças no projeto de um sistema antes que este seja implementado, com apenas uma fração do custo que se teria caso o sistema tivesse sido implementado;

• a credibilidade frente aos responsáveis pela tomada de decisões, visto que recursos, como animação, podem ser utilizados para ilustrar quão o modelo se aproxima da realidade;

• a versatilidade de um mesmo conjunto de métodos de simulação poder ser utilizado para analisar diversos sistemas.

Entretanto, em sistemas com elevado grau de complexidade são necessárias técnicas especiais para viabilizar a análise por simulação já que os modelos em questão também são relativamente complexos. Nestes casos, maior ênfase tem sido dada para simulação paralela e simulação distribuída. Na simulação paralela a execução dos modelos é feita em computadores paralelos. Já na simulação distribuída a execução dos modelos é realizada em computadores dispersos geograficamente, conectados através de redes de comunicação tipo LAN (local area network) ou WAN (wide area network), criando assim um "grande computador" que só existe no espaço virtual. Nas duas abordagens, são usados vários processadores, que participam no processo de simulação (FUJIMOTO, 1999; BANKS et al., 2000).

Dentre as razões para o uso dessas técnicas destacam-se:

• reduzir o tempo de execução dos modelos de grande porte e complexidade;

• explorar recursos computacionais apesar de estarem dispersos geograficamente;

• explorar a integração de ferramentas de modelagem e simulação, instalados em máquinas de diferentes fabricantes;

• assegurar redundância de recursos visando maior tolerância à falhas.

2.1. Simulação distribuída, baseada em rede de Petri

Modelos baseados em sistemas a eventos discretos (SEDs) são intensamente usados para descrever, analisar e controlar processos em sistemas produtivos, como por exemplo em ambientes de rede de computadores e manufatura, pois o que caracteriza este processo é o fato de sua dinâmica ser governada pela ocorrência de eventos instantâneos. Assim, a evolução de estados destes sistemas é baseada em regras que definem as condições para a ocorrência

de eventos e o resultado da ocorrência destes eventos (CASSANDRAS; STRICKLAND, 1992).

A rede de Petri (RdP) é uma técnica de modelagem gráfica e matemática originalmente desenvolvida para caracterizar operações de concorrência em computadores (MOORE; BRENNAN, 1996). Desde então ela tem sido utilizada para a modelagem de sistemas dinâmicos em diversas áreas (SRINIVASAN; VENKATASUBRAMANIAN, 1998), como por exemplo: protocolos de comunicação, algoritmos distribuídos, arquiteturas de computadores, interação homem/máquina (ELKOUTBI; KELLER, 1998).

Detalhes do funcionamento da rede de Petri (RdP) podem ser encontrados em MIYAGI, 1999, mas por não fazer parte do escopo do presente texto eles são aqui omitidos, isto é, o foco aqui é a simulação distribuída com RdP, por isso, só serão apresentadas aqui as características que fazem da RdP um boa ferramenta para modelar, analisar e controlar sistemas produtivos concorrentes, assíncronos, distribuídos e paralelos (INAMASU, 1995).

As principais características da RdP, segundo INAMASU (1995) são:

- permitir a representação de diferentes tipos de sistemas;
- ser um modelo que pode ser formalizado matematicamente;
- ser um modelo gráfico de aprendizado relativamente fácil, funcionando como linguagem de comunicação entre especialistas de diversas áreas;
- permitir a representação de paralelismo e sincronização;
- representar aspectos estáticos e dinâmicos;
- possuir métodos e ferramentas de análise, inclusive comerciais.

MOORE e BRENNAN (1996) apresentam mais algumas características da RdP:

- pode-se considerar regras de tempo associadas com [1]transições para representar o tempo requerido para completar uma atividade. Esta regra pode ser estocástica, baseada em uma função de probabilidade, um valor computado, ou uma constante.

  Regras de decisão podem ser associadas com os lugares e resolvem casos onde mais de uma transição está habilitada pela mesma marca ou conjunto de marcas (transições em conflito);

[1] Termos específicos da rede de Petri (como transição, lugar, marca, arco habilitador, arco inibidor) estão com letra "Times New Roman" tamanho 12.

- pode-se considerar atributos nas marcas para especificar um conjunto de características associadas com a marca. O valor destes atributos pode ser mudado nas transições e podem ser passados para algoritmos externos e o resultado incorporado ao modelo de RdP;
- pode-se considerar outros tipos de arcos, que proporcionam uma lógica adicional à transição: o arco habilitador, que habilita a transição caso o lugar a que se refere esteja marcado, sem consumir a marca; e o arco inibidor, que desabilita a transição caso o lugar a que se refere esteja marcado.

Dentre as RdP propostas (conforme as necessidades dos sistemas modelados/estudados), a RdP orientada a objetos têm recebido atenção por parte dos pesquisadores. Existem diferentes enfoques para RdP orientadas a objetos, mas a que cobre a maior diversidade de características/conceitos de objetos é a OPN (Object Petri Net) proposta por LAKOS (1994, 1995).

A OPN apresenta uma completa integração entre os conceitos de RdP e orientação a objetos, resultando na possibilidade de se modelar diretamente sistemas com níveis arbitrários de atividade, bem como apresentar flexibilidade na especificação da interface dos objetos, sejam elas síncronas ou assíncronas.

A OPN possui também arcos habilitadores e inibidores que facilitam a definição de funções de acesso que provêem informações sobre o estado do objeto sem alterá-lo. A utilização destas funções pode facilitar significativamente a modularização dos projetos em RdP.

Ainda com relação à OPN, cada RdP pode ser definida como uma classe, que pode ser instanciada em um diferente número de contextos (inclusive como marcas). Desta forma, uma marca pode encapsular uma sub-rede.

Isso implica que a OPN suporta a possibilidade de múltiplos níveis de atividade na rede e o projetista é livre para decidir como várias atividades devem ser compostas (se um objeto em particular deve ser ativo (que requisita/invoca uma ação), passivo (que realiza uma ação/atividade solicitada por outro objeto) ou ambos) (JUNQUEIRA, 2006).

Já na computação paralela aplicada a RdP, cada processador trabalha com uma lista local de transições, lugares e arcos, isto é, uma sub-RdP local, ao invés de uma lista global contendo todos os elementos da RdP, porém, essa abordagem traz problemas aos projetistas quanto a maneira de tratar a interação entre as sub-RdP que compõe a RdP global.

Para solucionar esse problema muitos autores sugerem a modelagem hierárquica, por exemplo, a abordagem adotada por JUNQUEIRA (2006), onde o sistema complexo é dividido em sub-sistemas (e em sub-modelos) mais fáceis de se processar e validar e, onde existem regras específicas de estruturação e conexão entre as partes.

Na divisão dos sistemas em sub-sistemas, deve-se definir a forma de comunicação entre as partes do problema. Esta comunicação define tipos específicos de interface que no caso de modelos baseados em RdP podem ser de três maneiras distintas, (JUNQUEIRA, 2006): interface entre modelos por fusão de lugares, interface entre modelos por fusão de transições e interface entre modelos por arcos habilitadores.

Na interface entre modelos por fusão de lugares, dois lugares de modelos distintos se comportam como um só. Contudo, essa abordagem apresenta um problema quando a chamada de um modelo ocorre simultaneamente por dois outros modelos, isto é, não se define quem fez o chamado primeiro.

Na interface entre modelos por fusão de transições, usa-se procedimento e interpretação análogos da técnica acima, entretanto, essa técnica não apresenta problemas em caso de mais de uma chamada no mesmo modelo.

Na interface entre modelos por arco habilitador, usam-se arcos habilitadores para habilitar transições dos modelos interligados, de modo a garantir o fluxo correto das informações. Contudo, nessa abordagem, pode não ocorrer à alteração de um estado, pois a habilitação do próximo estado de disparo, não implica na inibição do estado anterior, ou seja, qualquer transição pode disparar primeiro.

Ainda, de acordo com trabalho de JUNQUEIRA (2006) a modelagem de sistemas, considerando a sua simulação distribuída, apresenta seis etapas distintas, como mostra a Figura 1.

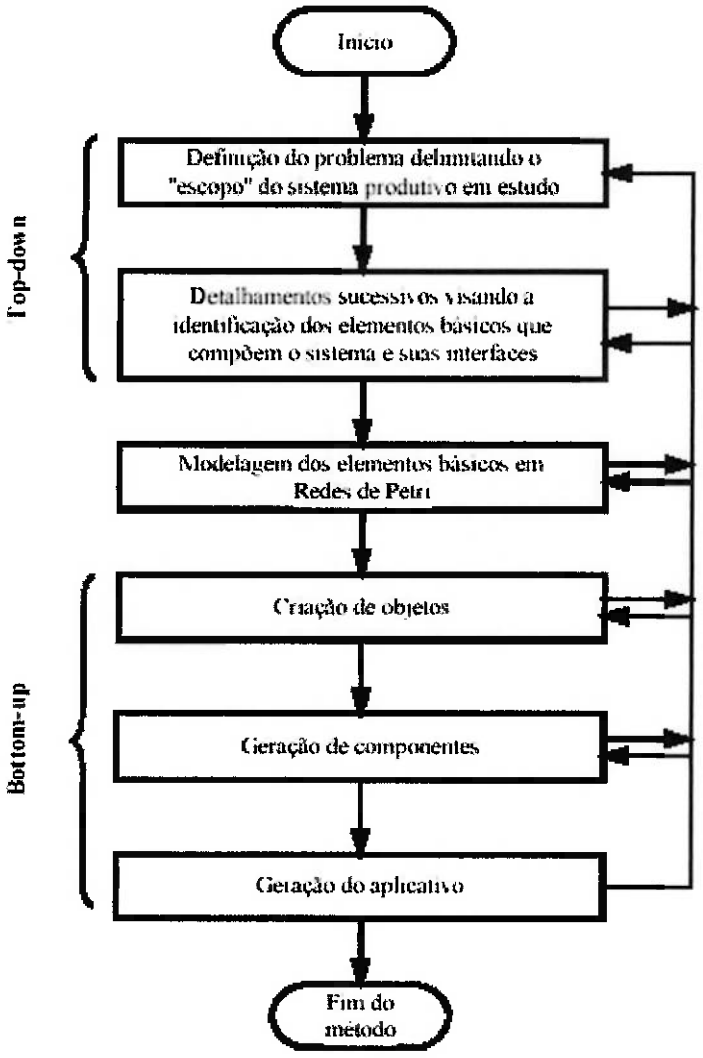

Figura 1. Diagrama do método de modelagem de sistemas.

Etapa 1 – Definição do problema delimitando o escopo do sistema produtivo em estudo

Nesta etapa deve-se delimitar o escopo do sistema produtivo em estudo, ou seja, quais departamentos, equipamentos e pessoas (ambos podendo ser vistos como recursos) são utilizados, quais as funcionalidades/características e processos que se pretende modelar e analisar (através de simulação).

Etapa 2 – Detalhamentos sucessivos visando a identificação dos elementos básicos que compõem o sistema e seus relacionamentos

Nesta etapa cada funcionalidade relacionada na etapa anterior deve ser detalhada até o nível requerido pelo modelista.

Etapa 3 – Modelagem dos elementos básicos em redes de Petri (RdP)

Nesta etapa as funcionalidades detalhadas no item acima são modeladas em RdP compondo o que é chamado de *classes*.

Etapa 4 – Criação de objetos

Aqui os *objetos* são gerados baseados no conceito de *classes*.

Etapa 5 – Geração de componentes

São gerados os *componentes* através da junção de *objetos* com suas interações próprias.

Etapa 6 – Geração do aplicativo

São gerados os *aplicativos* através da junção de *componentes*.

2.2. Ambiente de simulação distribuída

Uma simulação distribuída é constituída por processos lógicos que interagem via troca de mensagens, as quais possuem um registro de tempo associado. Um processo obedece ao relacionamento de causa e efeito, se e somente se cada processo lógico executa os eventos em ordem não decrescente do registro de tempo (FUJIMOTO, 1999). No presente caso, considera-se que os processos lógicos podem ser executados em computadores dispersos fisicamente.

Entre os meios de implementação de um ambiente de simulação distribuída, neste capítulo apresentam-se dois métodos que têm sido considerados para a comunicação entre computadores: *sockets* e CORBA.

2.2.1. Baseado em *sockets*

Uma das primeiras formas de se desenvolver aplicações distribuídas em um ou mais computadores foi através do uso de *sockets*. Com isso foram desenvolvidas diversas aplicações tipo cliente/servidor onde cliente(s) e servidor poderiam estar em computadores diferentes, distantes umas das outras. Atualmente existem outras ferramentas de linguagem para implementar software distribuído, mas é interessante notar como vários dos conceitos da API *(aplication program interface)* de *sockets* permanecem verdadeiros ainda hoje (Donahoo, 2001).

As APIs estão disponíveis para muitos sistemas operacionais, incluindo sistemas usados em computadores pessoais (por exemplo, Windows NT e Windows 98 da Microsoft) como também vários sistemas UNIX (por exemplo, Solaris da Sun Microsystems).

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. Deste modo, a API de *sockets* se tornou um padrão na indústria (The Java™ Tutorial, 2006).

Em se tratando de *hardware*, os *sockets* são como "encaixes" para o processador, isto é, como 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 de 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 (Donahoo, 2001), numa arquitetura de cliente/servidor.

A tecnologia cliente/servidor é uma arquitetura na qual o processamento da informação é dividido em módulos ou processos distintos. Um processo é responsável pela manutenção da informação (servidores) e outros responsáveis pela obtenção dos dados (os clientes).

Os processos cliente enviam pedidos para o processo servidor, e este por sua vez processa e envia os resultados dos pedidos.

Nos sistemas cliente/servidor o processamento tanto do servidor como o do cliente são equilibrados.

Geralmente, os serviços oferecidos pelos servidores dependem de processamento específico que só eles podem fazer. O processo cliente, por sua vez, fica livre para realizar outros trabalhos. A interação entre os processos cliente e servidor é uma troca cooperativa, em que o cliente é o ativo e o servidor reativo, ou seja, o cliente requisita uma operação, e neste ponto o servidor processa e responde ao cliente.

A Figura 2 mostra uma representação esquemática da utilização de *sockets*:

Figura 2 - Esquema representando a comunicação entre servidor-cliente via *sockets* e internet.

A programação dos *sockets* difere de um protocolo de I/O convencional porque um aplicativo deve especificar diversos detalhes para usar um *socket*. Por exemplo, um aplicativo deve escolher um protocolo de transporte em particular, fornecer o endereço de protocolo de uma máquina remota e especificar se o aplicativo é um cliente ou um servidor. Cada *socket* tem assim diversos parâmetros e, um aplicativo deve fornecer valores específicos para cada um deles (The Java™ Tutorial, 2006).

Para evitar que exista uma única função de *socket* com argumentos separados para cada opção, a API de *sockets* foi definida com muitas funções. 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).

No caso do compilador Microsoft Visual C++, a programação consiste na adição de um objeto [2]sock( ) (tanto para o programa cliente como para o programa

[2] Função específicas do protocolo de comunicação *sockets* estão com letra "Verdana" tamanho 12.

servidor) na definição de seus principais atributos e na confecção das subrotinas associadas a cada atividade de conexão (método).

Objeto cliente

O objeto cliente é aquele que deseja acessar alguma informação disponível no servidor (Donahoo, 2001).

Seus atributos elementares são:

servIP: geralmente utiliza-se o IP do servidor a se conectar.

servPort: determinação da porta onde será efetuada a conexão com a Internet via *sockets*.

Os principais métodos do objeto cliente são:

connect: atividades realizadas pelo cliente quando este se conecta com o servidor.

closesocket: atividades realizadas pelo cliente quando este se desconecta do servidor.

send: atividades realizadas pelo cliente quando o este envia informações para o servidor.

recv: atividades realizadas pelo cliente quando este recebe informações do servidor.

WSAStartup: carrega a biblioteca winsock, que nada mais é do que a biblioteca do windows que contém as funções do *sockets*.

Objeto servidor

O objeto servidor é aquele que gerencia a conexão com clientes por conter informações desejadas pelos mesmos (Donahoo, 2001).

Seu atributo elementar é :

servPort: determinação da porta onde será efetuada a conexão com a Internet via *sockets*.

Os principais métodos do objeto servidor são:

listen: atividades realizadas pelo servidor quando quando o cliente se conecta.

closesocket: atividades realizadas pelo servidor quando o cliente se desconecta.

accept: atividades realizadas pelo servidor quando o cliente envia informações.

send: atividades realizadas pelo servidor quando ele envia informação para o cliente.

WSAStartup: carrega a biblioteca winsock, que nada mais é do que a biblioteca do windows que contém as funções do *sockets*.

Seqüência de Chamadas de Procedimento com *Sockets*

Um cliente e um servidor devem selecionar um protocolo de transporte que suporte o serviço sem conexão ou um que suporte o serviço orientado à conexão. O serviço sem conexão permite a um aplicativo enviar uma mensagem a um destino arbitrário a qualquer momento; o destino não precisa concordar se ele aceita a mensagem antes da transmissão acontecer. Em contraste, um serviço orientado à conexão exige que dois aplicativos estabeleçam uma conexão de transporte antes que possam ser enviados dados.

Para estabelecer uma conexão, os aplicativos interagem com o software de protocolo de transporte em seus computadores locais, e os dois módulos de transporte trocam mensagens através da rede de comunicação. Após ambos os lados concordarem que uma conexão seja estabelecida, os aplicativos podem enviar dados (The Java™ Tutorial, 2006).

A Figura 3 mostra a seqüência de procedimentos com *sockets* que chamam o cliente e servidor no arquivo de teste que foi implementado e que têm seus códigos listados no Anexo A desse trabalho.

Nele, o servidor deve chamar a função listen (que abre a porta configurada e fica esperando por uma mensagem) antes que o cliente chame a função connect (que se conecta na porta indicada pelo servidor).

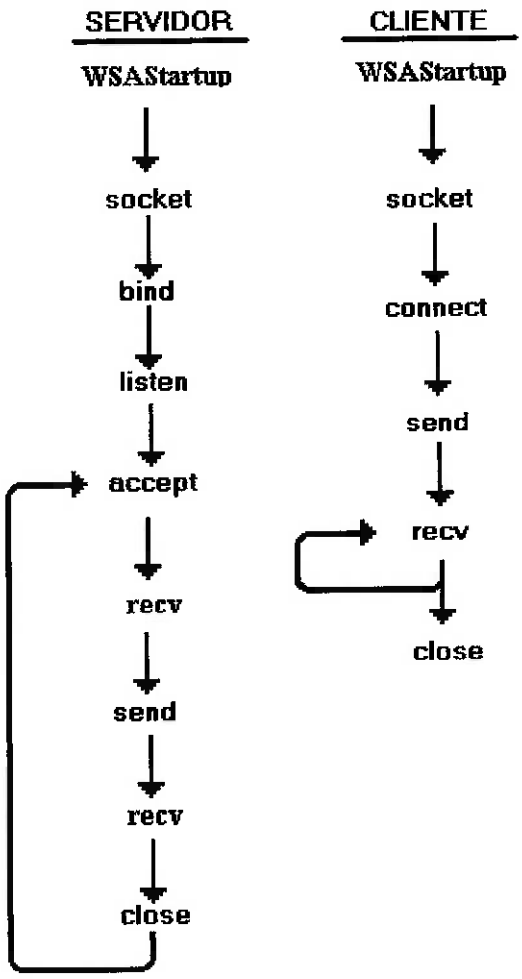

Figura 3 - A sequência de chamadas de procedimento com *sockets* no cliente e servidor exemplo.

Como mostra a Figura 3, o servidor chama nove procedimentos de *socket* e o cliente chama seis. O cliente começa chamando os procedimentos de biblioteca WSAStartup para carregar a biblioteca do *sockets* para windows. O cliente então chama *socket* para criar um *socket* e connect para conectar o *socket* a um servidor. Uma vez que é estabelecida a conexão, o cliente chama send para enviar os dados ao servidor e em seguida chama repetidamente a função recv para receber do servidor o eco do que foi recebido por ele.

Finalmente, após os dados terem sido recebidos, o cliente chama close para fechar o *socket*.

O servidor chama também WSAStartup para carregar a biblioteca do *sockets* antes de chamar *socket* para criar um *socket*. Uma vez que um *socket* foi criado, o

servidor chama bind para especificar uma porta de protocolo local para o *socket* e listen para colocar o *socket* em modo passivo. O servidor então insere um laço infinito em que ele chama accept para aceitar a próxima requisição de conexão recebida, recv para receber uma mensagem do cliente, send para enviar ao cliente o que foi recebido, novamente recv para avaliar se existe mais informação por receber e finalmente close para fechar a nova conexão. Depois de fechar uma conexão, o servidor chama accept para tratar a próxima conexão recebida.

2.2.2. Baseado em CORBA

CORBA (common object request broker architecture) é um padrão proposto por uma entidade chamada de OMG (object management group) cujo propósito é permitir interoperabilidade entre aplicações em ambientes distribuídos e heterogêneos. Este padrão estabelece a separação entre a interface de um objeto e sua implementação. Para descrição da interface do objeto, o CORBA oferece uma linguagem para definição de interfaces. Para implementação do objeto CORBA, pode ser utilizada qualquer linguagem de programação que tenha o binding (termo usado para denotar a chamada de um recurso (biblioteca) a partir de um programa para o qual esse recurso não é nativo) para CORBA (como XML por exemplo) (OMG, 2006).

A arquitetura CORBA é composta por um conjunto de blocos funcionais que usam o suporte de comunicação do ORB (object request broker) - o elemento responsável por coordenar as interações entre os objetos, interceptando as chamadas dos clientes e direcionando-as para o servidor apropriado (OMG, 2006).

Todo objeto CORBA possui uma identificação, chamada referência do objeto, que é atribuída pelo ORB na criação do objeto. Para usar um objeto, o cliente deve obter a sua referência, pois na invocação de um método sobre o objeto, o ORB o identifica através de sua referência (PANDOLFI, 2005).

O CORBA fornece as abstrações e os serviços necessários ao desenvolvimento de aplicações distribuídas e portáveis, sem a necessidade de que o programador se preocupe com detalhes de baixo nível. Ele fornece suporte para

vários modelos do tipo pedido-resposta facilitando a localização e a ativação transparente de objetos, e para a independência de linguagens de programação e de sistemas operacionais, o que propicia uma base sólida tanto para a integração de sistemas legados quanto para o desenvolvimento de novas aplicações distribuídas (OMG, 2006).

Vantagens:

A principal vantagem do uso de CORBA, e dos outros padrões OMA (object architecture model), é sua capacidade de permitir o desenvolvimento de aplicações que são interoperáveis em um ambiente de programação de alto nível. Isto é feito de maneira independente de sistema operacional, linguagem de programação, protocolos e topologia da rede de comunicação. CORBA utiliza recursos de orientação a objetos como polimorfismo, herança e encapsulamento, facilitando a implementação de componentes reutilizáveis de software. Esta arquitetura provê ainda um conjunto de serviços abertos e padronizados (*CORBAServices* e *CORBAFacilities*) que facilitam a implementação de aplicações distribuídas (Souza, 2006).

CORBA é empregado também na integração de sistemas legados, através da definição de interfaces IDL (interface definition language) para aplicações destes sistemas.

A IDL é independente de linguagem, suportando múltiplas "amarrações" de linguagem a partir de uma única especificação. Sendo assim, interfaces de software precisam ser escritas apenas uma vez.

A linguagem também é independente de plataforma. Uma interface assim especificada se mostra consistente com qualquer ORB e plataforma. Seu uso evita, portanto, problemas de portabilidade de plataforma, que podem ser provenientes de sistemas operacionais e interfaces definidas pelo fabricante.

Deve-se salientar, também, que a IDL é puramente uma especificação. Não há restrições quanto a implementações do objeto que define as interfaces, baseado em CORBA. Essas implementações podem utilizar qualquer linguagem e tipo de

algoritmo, que ficam escondidos dos clientes que utilizam a interface. Essa separação é uma das grandes vantagens da abordagem de CORBA, pois garante a reusabilidade e a redução da necessidade de construção de software.

Desvantagens

O CORBA apresenta um custo relativamente elevado de aprendizagem e utilização. CORBA é uma tecnologia que envolve conceitos avançados de programação orientada a objetos e passível dos problemas encontrados no uso de frameworks.

A compreensão dos conceitos de ORB e dos serviços de objetos com sua posterior utilização bem sucedida constitui um processo relativamente demorado e custoso de aprendizado, normalmente baseado em tentativas e erros (CORBA em C++, 2006).

A maioria das implementações de domínio público provêm apenas as funcionalidades básicas do ORB.

A integração de clientes CORBA com servidores implementados para ORBs de fabricantes diferentes é ainda uma tarefa não trivial. Isto deve-se principalmente a diferenças relacionadas aos padrões implementados por cada ORB (CORBA em C++, 2006).

3. SISTEMA MINICIM INSTALADO NO PMR-EPUSP

O MiniCIM (nome do sistema flexível instalado no PMR-EPUSP), é considerado neste trabalho como estudo de caso para comprovação das soluções propostas.

Dessa maneira, seus componentes foram estudados para que seu funcionamento fosse correto. Portanto, nesse capítulo é apresentado o sistema de funcionamento do MiniCIM, bem como seus componentes e seus respectivos modos de operação.

3.1. MiniCIM

O MiniCIM (*CIM - Computer Integrated Manufacturing*) (Rembold; Naji; Storr, 1994) da empresa Festo considerado neste trabalho é um sistema de manufatura integrado por computador que realiza um processo de montagem automática de diferentes peças (Festo, 1998). Cada peça é composta por: uma base (cilindro), de três cores possíveis (preta, prateada e rosa); um pino para sustentar uma mola, de dois tipos possíveis (preto ou cinza); a mola citada e uma tampa, ambos comuns aos três tipos. De acordo com a cor da base, as peças montadas assumem diferentes composições (Figura 4).

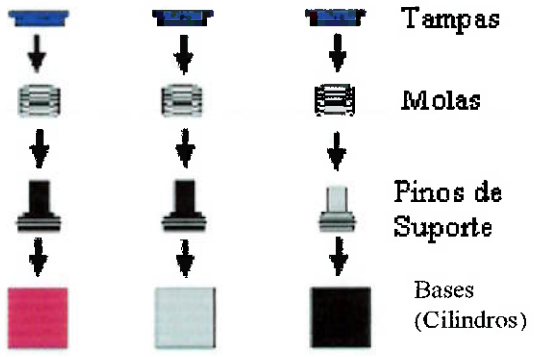

Figura 4 – Montagem dos diferentes cilindros pelo MiniCIM.

3.1.2. Descrição Geral

O MiniCIM é um equipamento para fins de treinamento especializado da empresa Festo que, na configuração disponível no PMR-EPUSP é composto de 5 estações de trabalho, cada uma capaz de ser operada individualmente ou, no contexto de um sistema integrado e automatizado, cada estação possui certa autonomia na execução de suas tarefas. As estações de trabalho são as seguintes: Estação de Testes; Estação de Distribuição; Estação de Montagem com Unidade de Execução da Montagem; Sistema Inteligente de Transporte (SIT); Estação de Controle de Célula de Trabalho (Figura 5).

A Estação de Distribuição armazena as bases (cilindros) e as encaminha ao processo de produção 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 (cilindros) vindas da estação de distribuição. Esta estação é também responsável por descartar peças rejeitadas nestes testes.

O Sistema Inteligente de Transporte (SIT) conta com uma esteira de transporte e cinco carros (*pallets*). Este sistema de transporte tem pré-definido alguns locais distintos para a parada dos carros, sendo um deles a Estação de Testes e outro a Estação de Montagem. O SIT destina-se a transportar as bases (cilindros) identificadas e testadas da Estação de Testes para a Estação de Montagem, onde a montagem da peça é finalizada. As peças montadas são então transportadas pelo SIT para um outro local previsto para a armazenagem das peças e liberação dos carros (*pallets*).

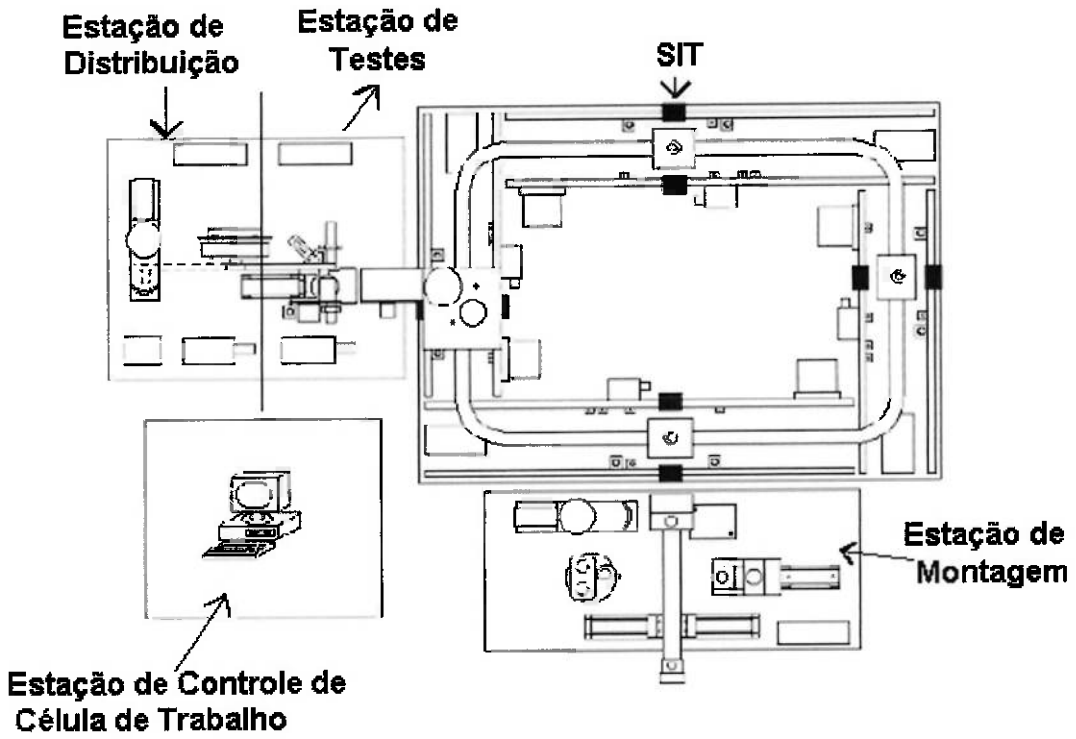

Figura 5 – Esquema ilustrativo do sistema MiniCIM.

3.1.3. Estação de montagem

A estação de montagem (Figura 6) é responsável pela confecção da peça final, por juntar todos os elementos que a constitui.

Para isso, a estação possui um manipulador que executa as tarefas, na seguinte ordem: recolhe do SIT o *pallet* com a base, pega o pino adequado para a base e o instala na base, coloca uma mola no pino, coloca a tampa e a fecha, e por fim recoloca o *pallet* de volta no SIT com a peça pronta.

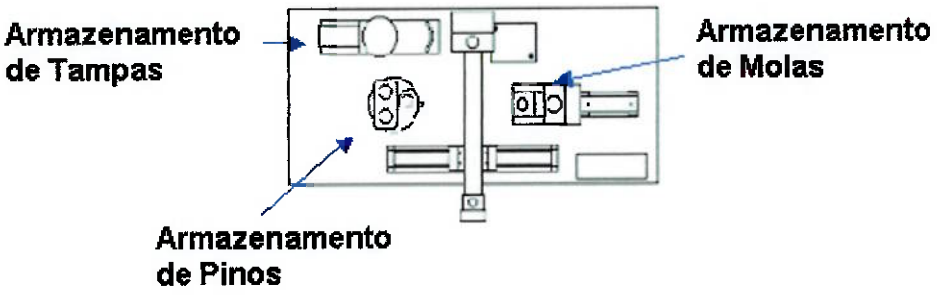

Figura 6 – Estação de montagem do MiniCIM.

3.2. Comunicação Profibus

O MiniCIM adota o Profibus para a rede de comunicação entre os CLPs que formam o sistema de controle da linha de produção.

3.2.1. História

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 comunicações complexas 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 orientadas, o Profibus começou seu avanço inicialmente na automação 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.

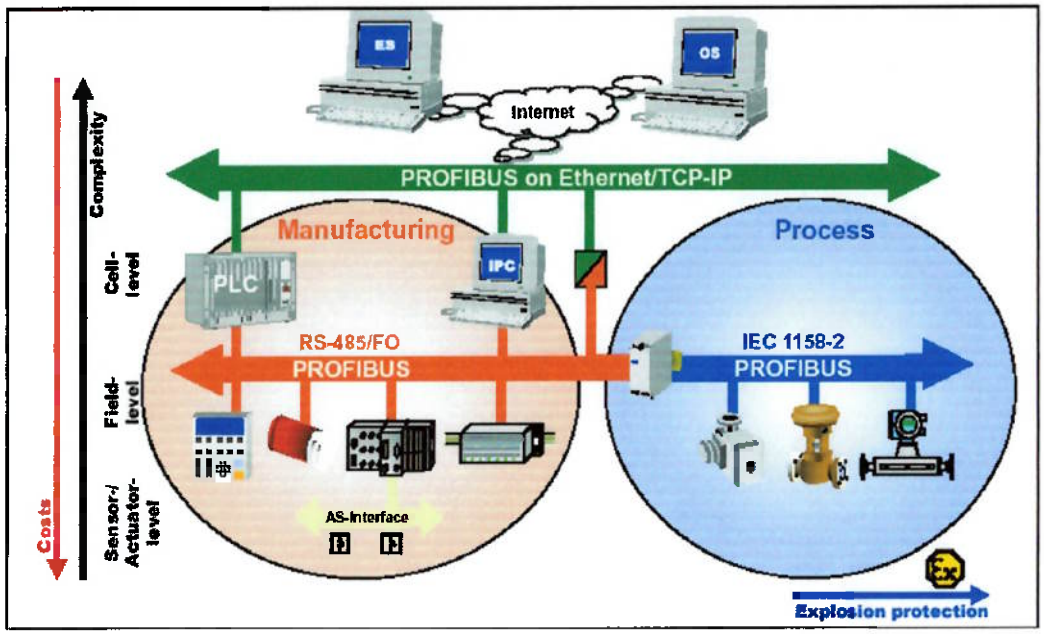

Figura 7 - Comunicação Industrial.

3.2.2. Profibus-DP - Periferia Descentralizada (Decentralized Periphery)

O Profibus especifica as características técnicas e funcionais de um sistema de comunicação industrial, através dos quais dispositivos digitais podem se interconectar, do nível de campo como os (CLPs) até o nível de células (como os sensores, atuadores, válvulas) (Figura 7).

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 à comunicação de dados no barramento. Um mestre pode enviar mensagens, sem uma requisição externa, sempre que possuir o direito de acesso ao barramento (o *token*). 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, acionamentos de velocidade variável 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 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.

3.2.3. 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 cobre blindado (*shieldado*) com um único par condutor 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, bem como uma colocação em funcionamento do tipo passo-a-passo, sem afetar outras estações. 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 (Figura 5). 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 Tipo-A, 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 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 or tipo Híbrido para IP65/67.

Tabela 2 - Distâncias baseadas em velocidade de transmissão para cabo Tipo A

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 Profibus são oferecidos por vários fabricantes. Uma característica particular é o sistema de conexão rápida.

Durante a instalação, deve-se observar atentamente a polaridade dos sinais de dados (A e B). O uso da blindagem é essencial para se obter 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 8.

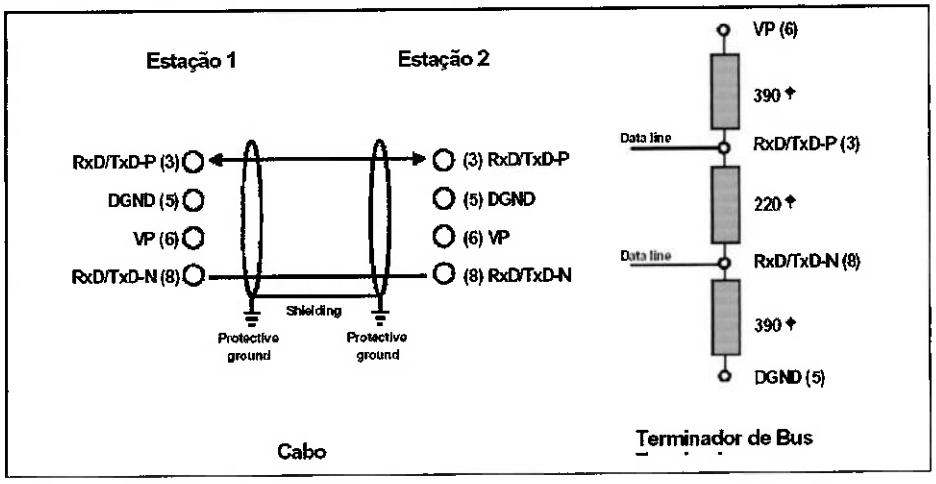

Figura 8 - Ligação e terminação para o RS-485

3.3. CLP estudado

Para o controle da estação de montagem do MiniCIM (apresentada na seção 3.1.3), é necessário o estudo de seu CLP, que no caso, é um Controlador Programável Siemens SIMATIC S7-300 com uma CPU 315-2 DP.

3.3.1. Controlador programável S7-300

O SIMATIC S7-300 (Figura 9) é um controlador modular amplamente utilizado em aplicações industriais centralizadas ou distribuídas de pequeno a médio porte.

Ele possui uma CPU que possibilita tempos de ciclo, 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	0,3 µs a 0,62 µs
• Operações com palavras	1µs
Faixas de endereçamento	
• Área total de endereçamento de E/S	1/1 Kbyte
• Área de Imagem de processo E/S	128/128 byte
• Canais digitais totais	máx. 8192 (1024 centrais)
• Canais analógicos totais	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 CLP 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 Figura 9 apresenta a estrutura do SIMATIC S7-300. É ela:

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.

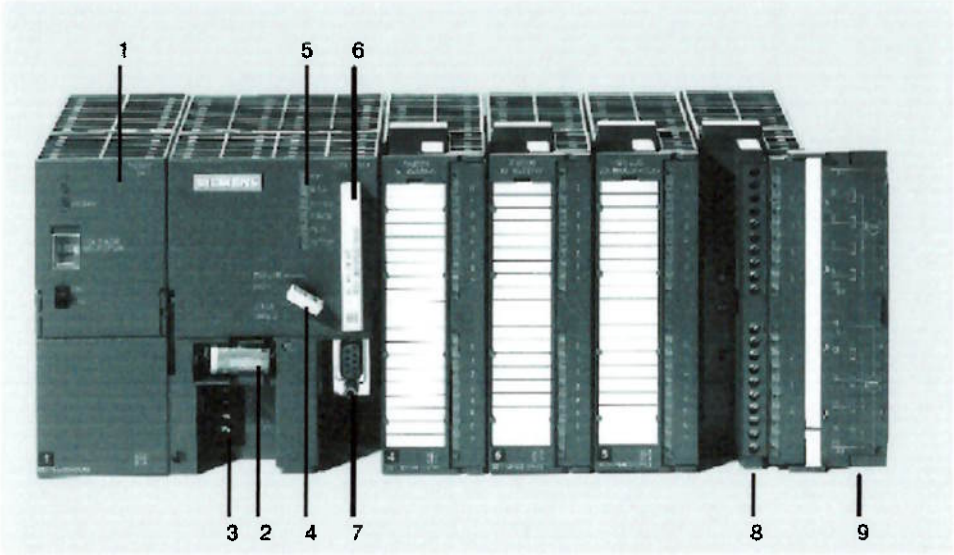

Figura 9 - SIMATIC S7-300 design.

A programação do CLP SIMATIC S7-300 é realizada com apoio do software STEP7 disponível no laboratório do PMR-EPUSP na versão SIMATIC STEP 7 Professional.

O STEP 7 Professional dispõe de recursos para tratar todas as linguagens de programação contempladas na norma internacional IEC 61131. São elas: ladder

(LAD), diagrama de blocos de função (FBD) e lista de instruções (STL), complementadas por três ferramentas:

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

4. AMBIENTE PROPOSTO

Neste capítulo é descrito o ambiente de simulação distribuída proposto para execução dos modelos a fim de verificar se tais sistemas atendem os objetivos para os quais foram propostos.

A estrutura geral da parte de controle e supervisão de um sistema produtivo pode ser dividida em três níveis hierárquicos conforme indicado na Figura 10. No nível diretamente relacionado com os sinais de atuadores e sensores e respectivos dispositivos de controle o foco está na comunicação e programação dos controladores. Um conjunto destes controladores programáveis estão conectados, em geral, através de algum protocolo de comunicação padrão (como por exemplo o Profibus na área industrial), a um supervisor 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, estabelecendo com isto a devida integração dos gerenciadores de plantas que compõem o sistema produtivo independente de onde elas de fato, estão fisicamente/geograficamente instaladas.

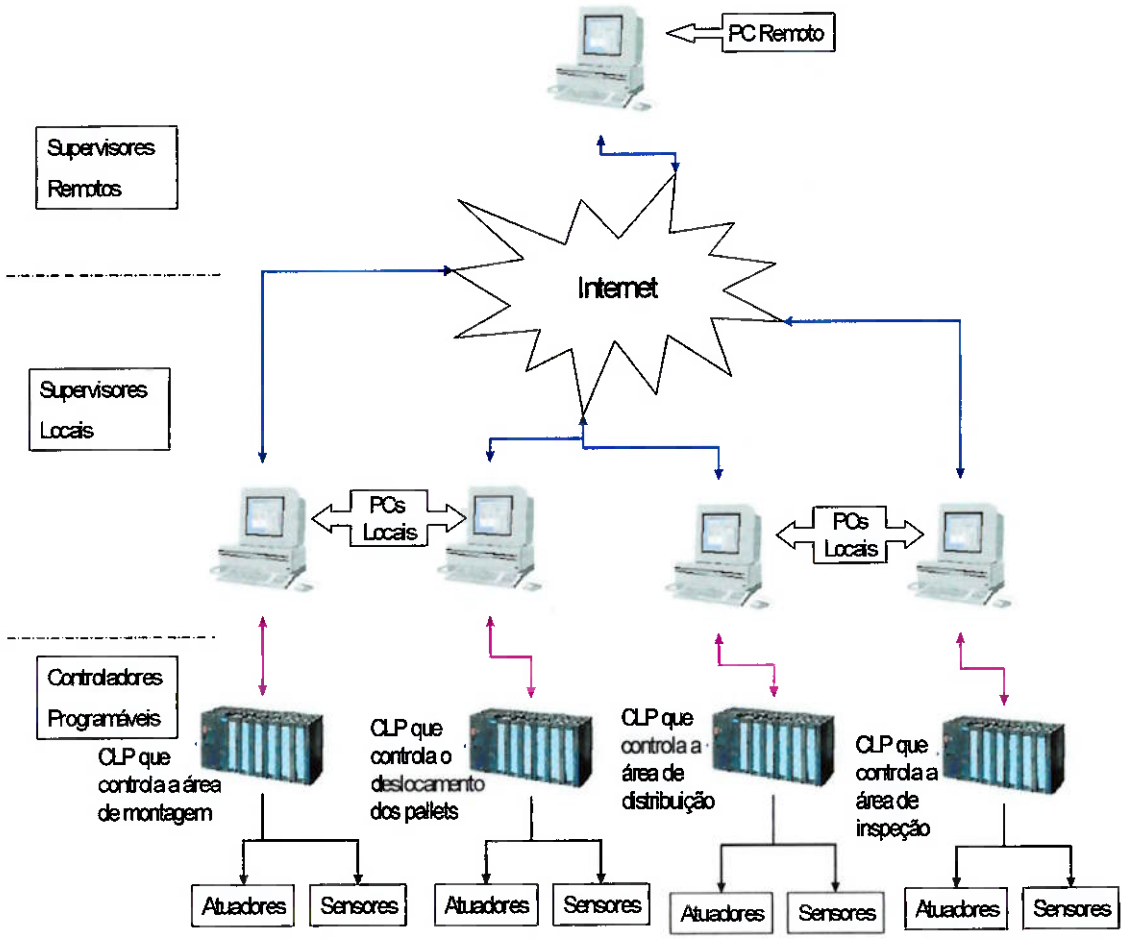

Figura 10 – Ambiente proposto para simulação distribuída no sistema flexível do PMR-EPUSP.

Considerando como um estudo de caso, um sistema flexível de montagem, citado no capítulo 3, a seguinte simplificação foi adotada:

- 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.

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 servidor, que por sua vez é controlado pelo servidor central do sistema.

Na comunicação entre os supervisores locais e remotos através da internet, foram estudados duas propostas: uma via *sockets* e outra via *middlewares* (por exemplo CORBA) e, com base no trabalho de Pandolfi (2005), foi desenvolvida uma implementação via *sockets*.

Observa-se ainda que dentre as várias estações de trabalho do sistema flexível de montagem considerado, a implementação aqui foi desenvolvida apenas para a estação de montagem que envolve tarefas de maior complexidade.

Quanto ao hardware, propõe-se a instalação de um computador para cada CLP do sistema produtivo, para que cada área tenha seu próprio servidor. Assim, cada parte do sistema pode funcionar de modo independente do resto.

O uso de webcams também se faz necessário para que as imagens do sistema produtivo sejam enviadas do computador local para o operador que está no computador remoto. Desta maneira, o monitoramento se torna mais eficaz.

4.1. O ambiente implantado

Para a implementação nesta etapa, o software desenvolvido para atender a lógica do sistema tem sua programação dividida em três partes distintas feitas em C++ com interface gráfica em plataforma Windows e arquitetura cliente/servidor. Cada uma dessas partes tem uma função específica:

- a primeira parte diz respeito a "abertura" de comunicação com o CLP, com envio e recebimento de informação (programa servidor que fica no PC Local). Essa comunicação entre o computador local (que emula o servidor) com o CLP da área de montagem, não foi desenvolvida através do protocolo Profibus e sim pelo protocolo serial;
- a segunda parte é a comunicação via internet usando o artifício do *sockets* para abertura e comunicação dos computadores que participam do ambiente (tanto o programa cliente como o programa servidor possuem as funcionalidades do *socket*).
- e a terceira parte se refere a programação do sistema produtivo, ou seja, a construção de um programa de controle em código C++, criou-se uma linha de produção que simula o funcionamento da área de

montagem. Contudo, é importante destacar que esse programa não estará dentro do CLP e sim do supervisor remoto (programa cliente que fica no PC remoto).

Esta implementação de teste, para comprovação da comunicação existente no ambiente proposto, tem a estação de montagem do sistema produtivo MiniCIM controlado e supervisionado remotamente (Figura 11).

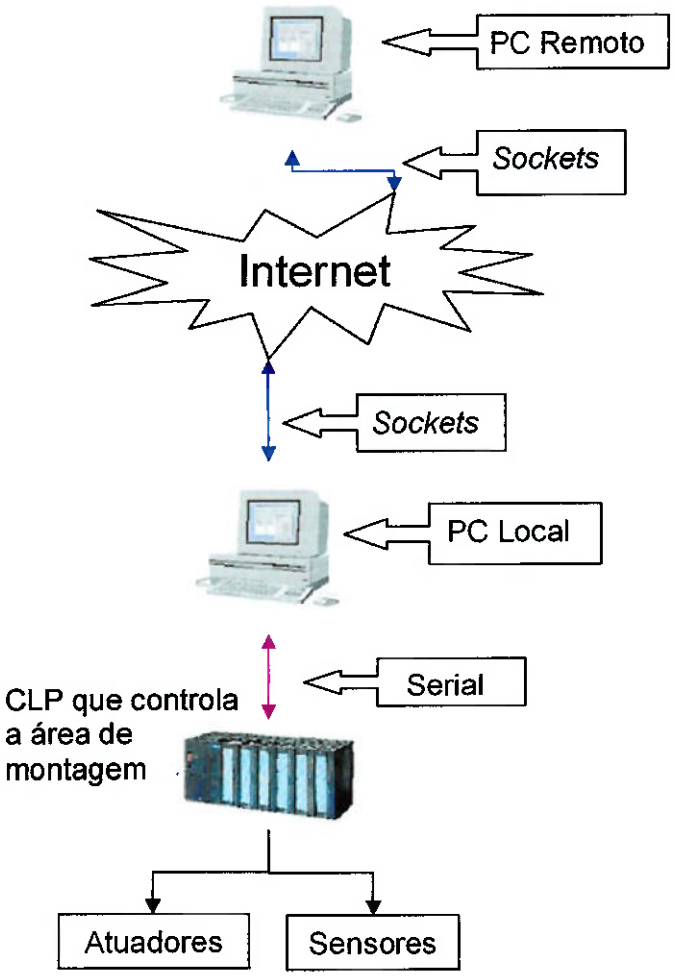

Figura 11 – Implementação teste para um ambiente de simulação distribuída.

A seguir é apresentada cada uma das partes do ambiente que foi implementado.

4.1.1. "Abertura" de comunicação serial

Para a primeira etapa desse software uma comunicação serial com o CLP S7-300 foi implementada na parte do servidor. Foram utilizados os recursos do Sismatic S7-Prodave 6.0 da Siemens, que disponibiliza duas bibliotecas para serem utilizadas no programa em C++. As bibliotecas em questão tem seus códigos listados no Anexo B, assim como o código dos programas cliente e servidor do ambiente implementado.

Tais bibliotecas, depois de referenciadas corretamente no programa, foram usadas para "abrir" a comunicação com o CLP através do cabo serial e definir os sinais nos conectores de saída do CLP, assim como ler os sinais nos conectores de entrada do CLP. Observa-se que foi utilizado um conector RS-232 entre o computador e o CLP, para transmitir os pacotes de dados do computador local para os conectores devidos do CLP.

O programa servidor, que fica instalado no computador local, e que tem sua tela de entrada apresentada na Figura 12, tem a função de "abrir" a comunicação com o CLP através da comunicação serial, invocando a função load_tool( )[3] e passando os parâmetros de conexão que são: número da conexão a ser criada, o endereço da porta serial, a identificação adotada para essa conexão, o número do rack (posição da CPU do CLP no trilho de montagem, geralmente 0) e por fim o número do slot (endereço da CPU, geralmente 2) da mesma.

Ao clicar em [Abir] a função de abertura de conexão é executada sendo possível visualizar o sucesso da mesma pelo led de conexão do conector RS-232 que muda de estado (fica On).

O programa principal fica constantemente lendo os sinais de saída e entrada do CLP nos endereços indicados e quando requisitado pelo cliente, atualiza os sinais de saída.

[3] Funções específicas do programa implementado (como load_tool( ), a_field_write( ), e_field_read( ) e a_field_read( )) estão com letra "Tahoma" tamanho 12.

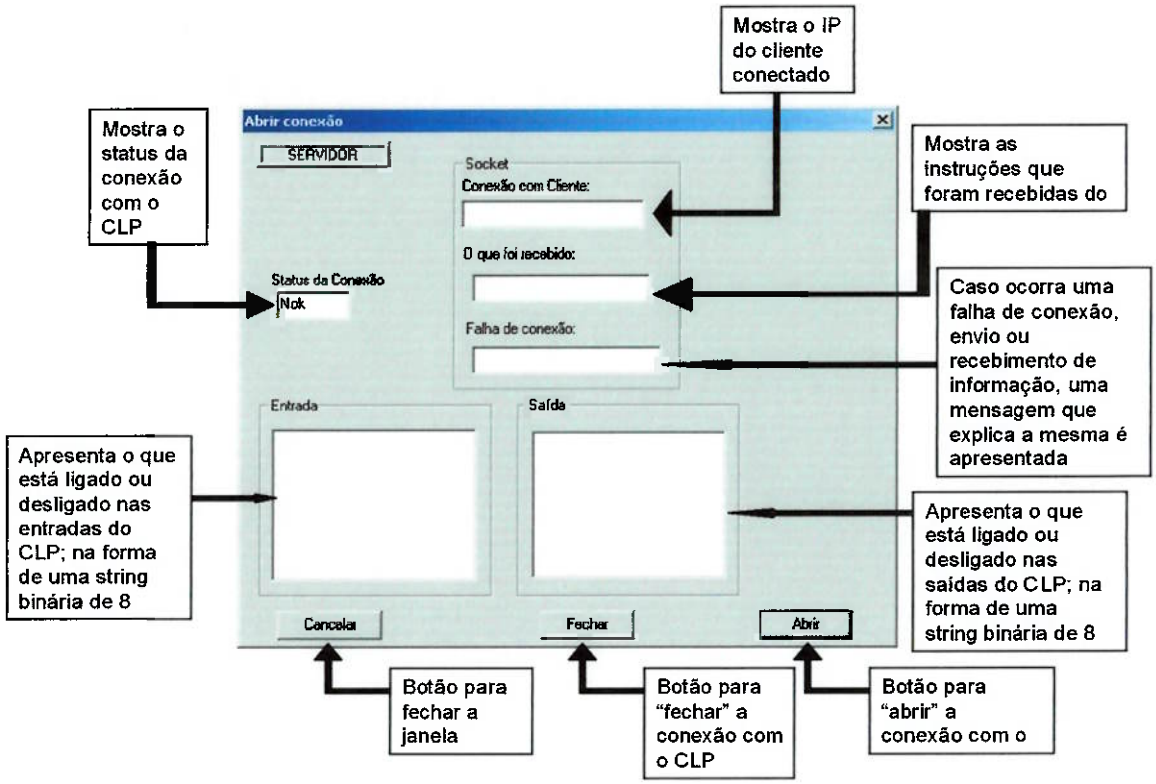

Figura 12 – Tela do servidor.

Para interagir com o CLP, as funções a_field_write( ), e_field_read( ), a_field_read( ) foram usadas para atualizar os sinais de saída, ler os sinais nas entradas e ler os sinais nas saídas respectivamente. É importante ressaltar que o processo de leitura e o processo de escrita estão relacionados com o hardware do CLP, isto é, os módulos físicos/cartões. No caso de um cartão de oito entradas digitais e oito saídas digitais, por exemplo, a leitura, ou entrada de dados/sinais deve ser no formato 00000000 em binário (caso esteja tudo desligado).

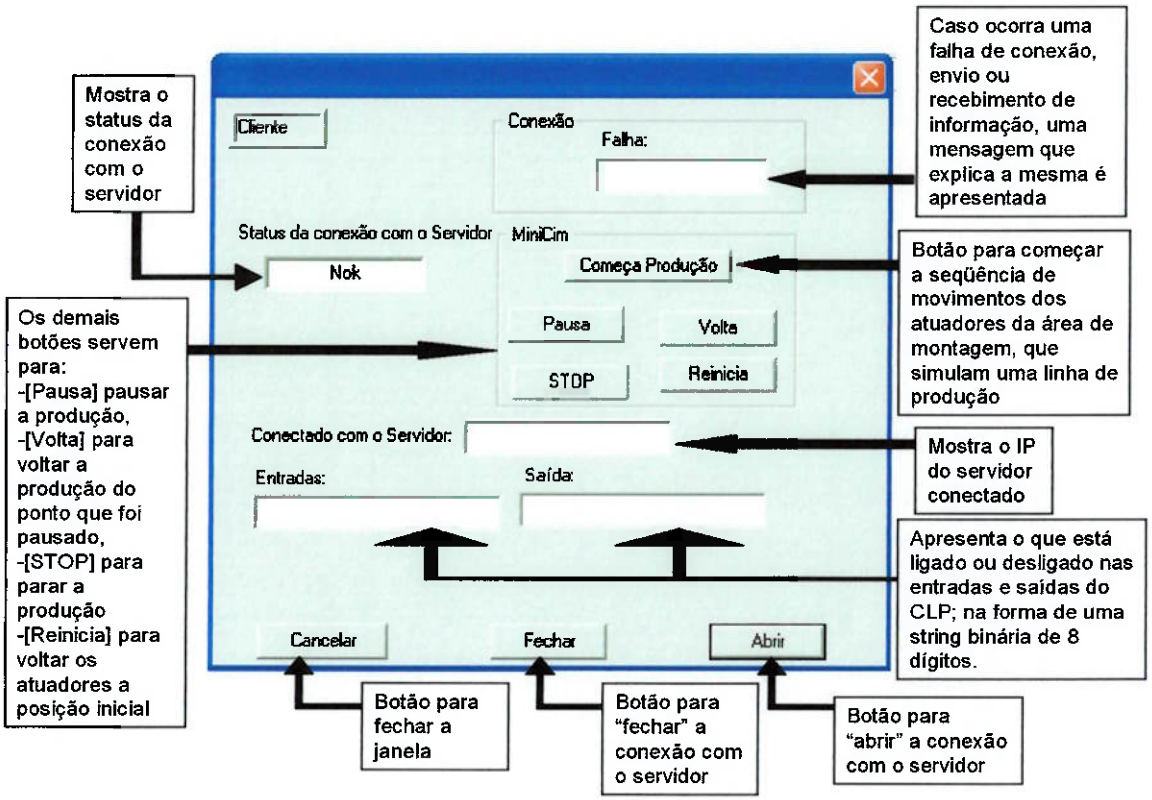

Figura 13 - Tela do cliente.

A tela do programa cliente, que fica instalado no computador remoto, é apresentada na Figura 13. A programação do cliente não contempla uma comunicação serial. Ele tem apenas a implementação em *sockets* e os comandos para o controle do MiniCIM.

Ao clicar no botão [Abrir], o cliente abre a conexão com o servidor e fica constantemente recebendo as leituras da entrada e da saída do CLP.

Na parte dos comandos para o MiniCIM, existem os botões [Começa Produção], [Pausa], [Reinicia] e [STOP]. Com esses quatro botões é possível comandar uma linha de produção que é apresentada na seção seguinte.

4.1.2. Testes do sistema

Para avaliar o sistema implementado, alguns testes foram conduzidos no MiniCIM instalado no PMR-EPUSP.

Os testes envolveram procedimentos de comando e monitoração relacionados com a ativação de atuadores e coleta de dados de sensores instalados na estação de montagem, descrita na seção 3.1.3.

Um exemplo da seqüência de comandos é demomstrado na Tabela 5, onde os sinais dos cartões de memória do CLP são apresentados. Por exemplo, para descer o braço do manipulador na direção de Z, deve-se enviar para o cartão de memória de endereço 16 a string binária 00000100, acionando então o motor do eixo.

A seqüência de comandos fica configurada no programa cliente. É do cliente que parte a requisição para que o servidor execute a operação. Portanto, toda e qualquer implementação específica da atividade a ser desenvolvida, deve ser criada pelo cliente.

Essa abordagem, de se ter a atividade no programa cliente, é de grande vantagem num sistema distribuído, o qual muitas vezes se mostra disperso entre em os continentes, pois dessa maneira, no caso de correções ou alterações os clientes não precisam se deslocar até o lugar do servidor e ainda, o mesmo servidor atua como operador para inúmeros clientes diferentes.

Tabela 5 – Passos da linha de produção do estudo de caso.

Passos para a Produção									
Passos do Movimento	**.0**	**.1**	**.2**	**.3**	**.4**	**.5**	**.6**	**.7**	**Endereço do cartão de memória**
Desce o braço do manipulador na direção de Z	0	0	0	0	0	1	0	0	16
Para o braço do manipulador	0	0	0	0	0	0	0	0	16
Gira a garra no sentido horário	0	0	1	0	0	0	0	0	16
Fecha a garra	1	0	0	0	0	0	0	0	16
Avança o pistão que libera a tampa do buffer	0	0	0	0	0	1	0	0	20

Retira um pino preto do buffer	0	1	0	0	0	0	0	0	20
Recolhe o pistão que retira a mola do buffer	0	0	1	0	0	0	0	0	20
Recolhe o pino que prende a base no meio da área de montagem	0	0	0	0	0	0	0	1	20
Avança o pino que prende a base no meio da área de montagem	0	0	0	0	0	0	1	0	20
Avança o pistão que retira uma mola do buffer	0	0	0	0	0	0	0	0	20
Retira um pino prata do buffer	1	0	0	0	0	0	0	0	20
Recolhe o pistão que liberou uma tampa do buffer	0	0	0	0	1	0	0	0	20
Abre a garra	0	1	0	0	0	0	0	0	16
Gira a garra no sentido anti-horário	0	0	0	1	0	0	0	0	16
Sobe o braço do manipulador na direção de Z	0	0	0	0	1	0	0	0	16
Para o braço do manipulador	0	0	0	0	0	0	0	0	16

4.1.3. Comunicação via internet

Para a execução da comunicação via internet, foi usado o artifício do *sockets*, conforme foi estudado anteriormente neste trabalho.

No programa do servidor e do cliente, ao clicar no botão [Abrir] (Figura 12 e Figura 13), os programas abrem uma conexão via *socket* que, no caso do servidor, fica esperando um cliente se conectar e no caso do cliente, fica esperando um comando para enviar.

Para que o programa servidor não fique apenas esperando a conexão do cliente, a função de accept do *socket* fica numa thread, que é executada em paralelo ao programa principal e que, quando recebe a conexão, envia essa informação para o programa principal que recebe os comandos e os executa.

4.1.4. Discussão dos testes realizados

Durante os testes realizados na estação de montagem, atuadores foram acionados e estados de sensores foram lidos de um computador remoto. Para isso, foi implementada uma comunicação serial entre o computador local e o CLP da estação de montagem, que funcionou sem atrasos e sem interrupções.

No caso da comunicação via internet, os tempos de resposta também foram satisfatórios, sendo, portanto, quase imediatas às respostas dos atuadores da estação de montagem aos comandos dados no PC remoto.

Entretanto, a visualização das imagens da estação de montagem (Figura 14), enviadas pela webcam instalada junto ao computador local para o computador remoto, ainda tem um delay muito grande. Contudo, espera-se que a próxima geração da internet diminuia esse delay de maneira significativa.

Figura 14 – Visualização da estação de montagem pela webcam.

5. CONCLUSÂO

Para comprovação do ambiente de simulação distribuída proposto, foi considerado um sistema flexível de montagem por se tratar de um sistema produtivo modular, ele serve ao propósito de se estudar cada parte individualmente para posteriormente estudá-las como um único sistema.

Contudo, para a análise completa do ambiente proposto, é necessário que as outras estações desse sistema produtivo estejam operando, como mostrado na Figura 10, para que a interação entre os supervisores locais e os supervisores remotos possa ser avaliada.

Quanto à comunicação, o estudos realizados do protocolo Profibus deve ser implementados na continuidade desse trabalho, quando todo o sistema produtivo considerado for colocado no ambiente de simulção.

Já para a comunicação via internet, foram consideradas duas abordagens: *sockets* e CORBA.

O CORBA apesar de fornecer várias facilidades, como objetos independentes de linguagem e plataforma, apresenta uma relativa dificuldade de programação por tratar-se de um formato de programar de aprendizado não trivial.

Já o *sockets* se mostrou a melhor opção para o uso no projeto, pois tem um meio de programação mais acessível em C++, como demonstrado no presente texto, e já teve sua eficiência comprovada em trabalhos anteriores com o MiniCIM. Por isso, o *sockets* foi o protocolo escolhido.

Considera-se também, para o futuro desse trabalho, o advento da internet 2 que é a próxima geração da internet, tendo uma velocidade de 2,5 Gbps (bilhões de bits por segundo). Dessa maneira, o ambiente proposto trabalharia em tempo real, não importando a distância dos supervisores locais entre si e com o supervisor remoto.

Considerando-se que os principais conceitos foram assimilados através dos estudos realizados, e que o ambiente de simulação distribuída implementado foi testado com sucesso tanto na comunicação local como na comunicação pela internet.

Por fim, vale ressaltar a importância do estudo inicial, pois através dele identificou-se os parâmetros para os ambientes de simulação distribuída baseada em RdP, que envolvem a interface entre modelos por fusão de transições e a modelagem hierárquica para simulação de sistemas complexos, bem como o fechamento do escopo do caso de uso que foi aplicado apenas na estação de montagem do MiniCIM, através de comunicação serial e não por Profibus, visto que tal tarefa seria relativamente complexa para ser concluída ainda este ano.

5.1. Trabalhos futuros

Os estudos e pesquisas realizados sobre a comunicação Profibus serão posteriormente considerados na continuidade desse projeto, quando o ambiente de simulação distribuída proposto for desenvolvido com o controle de todas as estações do MiniCIM.

A incorporação das imagens da webcam na interface gráfica do cliente, também será considerada na continuidade do projeto.

6. REFERÊNCIA BIBLIOGRÁFICAS

BANKS, J. (Chair) *Simulation in the future.* In. **Proceedings of the 2000 Winter Simulation Conference**, p.1568-1576, 2000.

CASSANDRAS, C. G., STRICKLAND, S. G. ***Sample Path Properties of timed Discrete Event Systems in Discrete Event Dynamic Systems – Analizing Complexity and Performance in the Modern World.*** New York: IEEE Press, 1992.

CENTENO, M. A. *An Introduction to Simulation Modeling.* In: **Proceedings of the 1996 Winter Simulation Conference**, pp.15-22, 1996.

DONAHOO, M. TCP/IP *Sockets*, The C version, 2001.

ELKOUTBI, M., KELLER, R. K. *Modeling Interactive Systems with Hierarchical Colored Petri Nets.* In: **Proceedings of the 1998 Advanced Simulation Technologies Conference**, p.432-437, Apr. 1998.

EYKHOFF, P. ***System identification: parameters and state estimation.*** London: John
Wiley, 1974.

FESTO. Modulares Produktions-Sistem Station Prufen, Lernsystem Automatisierung und Kommunkation. Esslingen: Festo Didactic Gmbh & CO, 1998.

FUJIMOTO, R. M. *Parallel and distributed simulation.* In. **Proceedings of the 1999 Winter Simulation Conference**, p.122-131, 1999.

INAMASU, R.Y., **Modelo de FMS: Uma Plataforma para Simulação e Planejamento**, 1995, 134p. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo. São Carlos, 1995.

JUNQUEIRA, F. *Modelagem e simulação distribuída de Sistemas Produtivos. Tese de Doutorado, Escola Politécnica da USP*, 2006.

LAKOS, C. A., *The Object Orientation of Object Petri Nets.* In: **Workshop on Object Oriented Programming and Models of Concurrency**, 1995.

LEE, W. B., LAU, H. C. W. Multi-agent modeling of dispersed manufacturing networks. **Expert Systems with Applications**, No. 16, p.297-306, 1999.

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

MOORE, K. E., BRENNAN, J. E. *Alpha/SIM Simulation Software Tutorial.* In: **Proceedings of the 1996 Winter Simulation Conference**, p.632-639, 1996.

PANDOLFI, C. *Comunicação entre Programas para a Simulação Distribuída de Sistemas Produtivos. Trabalho de Formatura, Dep. Eng. Mecânica, Escola Politécnica da USP*, 2005.

REMBOLD U.; NAJI, B.; STORR, A. **Computer Integrated Manufacturing And Engineering**. Londres: Addison-Wesley, 1994.

SHI, Y., GREGORY, M. *International manufacturing networks – to develop global competitive capabilities.* **Journal of Operations Management**, No. 16, pp.195-214, 1998.

SRINIVASAN, R., VENKATASUBRAMANIAN, V. *Automating HAZOP analysis of batch chemical plant: Part I. The knowledge representation framework.* **Computers Chem. Enginnering**, Great Britain, Vol. 22, No. 9, p.1345–1355, 1998.

Middleware – Uma abordagem às características e funcionalidades de CORBA. Disponível em: <http://www.inf.ufrgs.br/gpesquisa/procpar/disc/inf01008/trabalhos/sem00-1/CORBA/CORBAgeyer2.htm> Acesso em 5 de junho de 2006.

Palestra sobre introdução ao CORBA. Disponível em: <http://www.dicas-l.com.br/cursos/corba.ppt>. Acesso em 5 de junho de 2006.

Souza, I., Vantagens da utilização do CORBA no Gerenciamento de Redes. Disponível em: <www.lasid.ufba.br/eventos/wola99/resumos/ivone.html>. Acesso em 5 de junho de 2006.

OMG. Site oficial da OMG. Disponível em: <http://www.omg.org>. Acesso em 5 de junho de 2006.

CORBA em C++. Uma abordagem para o desenvolvimento de aplicações distribuídas em CORBA usando C++ e ORBIX. Disponível em: <http://www.ravel.ufrj.br/arquivosPublicacoes/helvecio_rel_corba.pdf>. Acesso em 5 de junho de 2006.

Profibus. Site oficial da organização mundial de usuários de Comunicação Profibus. Disponível em: <www.profibus.org>. Acesso em 1 de junho de 2006.

Siemens. Site da Siemens. Disponível em: <www.siemens.com>. Acesso em 1 de junho de 2006.

The Java™ Tutorial. Tutorial explicativo sobre a ferramenta *sockets*. Disponível em: <http://java.sun.com/docs/books/tutorial/networking/sockets/index.html>. Acesso em 1 de junho de 2006.

Anexo A: Teste de comunicação com *sockets*

Esta é uma implementação de um ambiente de comunicação usando *socket*, onde o servidor abre uma comunicação pela porta especificada e fica esperando por conexões de clientes infinitamente.

Quando o cliente se conecta, ele envia a frase "Teste de comunicação" para o servidor, que a recebe e envia a mesma frase para o cliente, como se fosse um eco (Figura 11).

Abaixo é demonstrada a listagem dos códigos do servidor e do cliente.

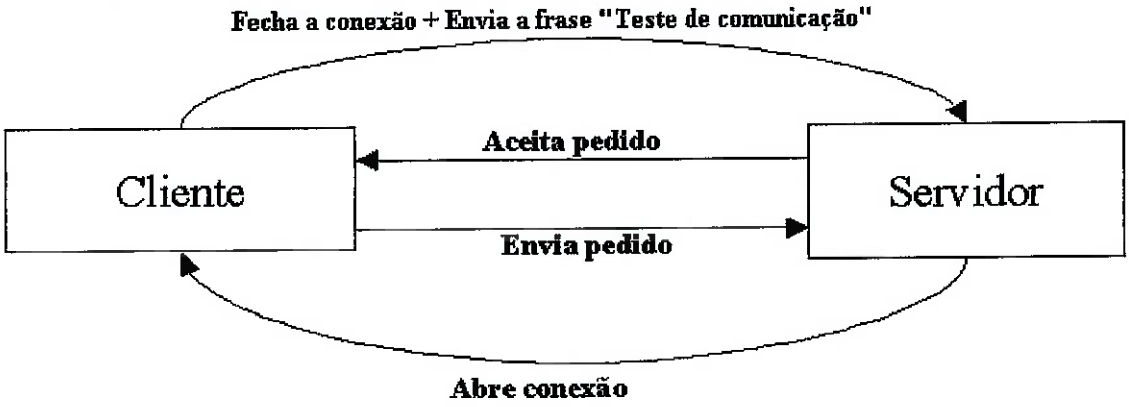

Figura 11 – Fluxograma de comunicação da implementação.

A.1 Arquivo do servidor:

```
#include <stdio.h>
#include <winsock.h>    // Biblioteca do sockets.
#include <stdlib.h>
#define RCVBUFSIZE 32   // Tamanho do buffer da mensagem recebida
#define MAXPENDING 5    // Máximo de conexões simultâneas.
void DieWithError(char *errorMessage);  // Função de erro.
void HandleTCPClient(int clntSocket);   // Função de TCP.
void main()                        // Laço principal.
{
    int servSock;                  // Descrição do Socket para o servidor
    int clntSock;                  // Descrição do Socket para o cliente
    struct sockaddr_in echoServAddr; // Endereço local
```

```
    struct sockaddr_in echoClntAddr; // Endereço do cliente
    unsigned short echoServPort;     // Porta do servidor
    int clntLen;            // Tamanho da estrutura de dados do endereço do cliente
    WSADATA wsaData;      // Estrutura para comunicação pelo WinSock
    echoServPort = 5600;  // Porta de acesso para o servidor
    if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0) // Carrega a winsock.dll 2.0
    {
      fprintf(stderr, "WSAStartup() failed");  // Falha no carregamento da
biblioteca
            exit(1);                                                    // Fecha o
aplicativo
    }
    // Cria um socket para conexão
    if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() failed");   // Falha na criação do socket
    // Constói a estrutura local de endereço
    memset(&echoServAddr, 0, sizeof(echoServAddr));   // Zera a estrutura de saída
    echoServAddr.sin_family = AF_INET;            // Família de endereços da
Internet
    echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); // Qualquer interface de
entrada
    echoServAddr.sin_port = htons(echoServPort);     // Porta Local
    // Bind para o endereço local
    if (bind(servSock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) <
0)
        DieWithError("bind() failed");
   // Marca o socket para escutar futuras conexões
    if (listen(servSock, MAXPENDING) < 0)
        DieWithError("listen() failed");
    for (;;) // Roda para sempre
    {
        // Seta o tamanho do parâmetro de entrada-saída
        clntLen = sizeof(echoClntAddr);
        // Espera pelo cliente se conectar
        if ((clntSock = accept(servSock, (struct sockaddr *) &echoClntAddr,
&clntLen)) < 0)
            DieWithError("accept() failed");
        // O socket do servidor está conectado ao cliente
        printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));
        HandleTCPClient(clntSock);
    }
}
void DieWithError(char *errorMessage)
{
    fprintf(stderr,"%s: %d\n", errorMessage, WSAGetLastError());
```

```
    exit(1);
}
void HandleTCPClient(int clntSocket)
{
    char echoBuffer[RCVBUFSIZE];        // Buffer para o string de eco
    int recvMsgSize;                    // Tamanho da mensagem recebida
    // Mensagem recebida pelo cliente
    if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
        DieWithError("recv() failed");
      printf(echoBuffer);
    // Manda a mensagem recebida e recebe de novo até o fim da transmissão
    while (recvMsgSize > 0)      // Zero indica o fim da transmissão
    {
        // Eco enviado de volta para o cliente
        if (send(clntSocket, echoBuffer, recvMsgSize, 0) != recvMsgSize)
            DieWithError("send() failed");
        // Verifica se tem mais dados para receber
        if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE,0)) < 0)
            DieWithError("recv() failed");
    }
    closesocket(clntSocket);    // Fecha o socket do cliente
}
```

A.2 Arquivo do cliente:

```
#include <stdio.h>
#include <winsock.h>   // Biblioteca do sockets
#include <stdlib.h>
#define RCVBUFSIZE 32  // Tamanho do buffer da mensagem recebida
void DieWithError(char *errorMessage);  // Função de erro.
void main()    // Laço principal
{
        int sock;    // Descrição do Socket para o cliente
         struct sockaddr_in echoServAddr;   // Endereço local
        unsigned short echoServPort;  // Porta do servidor
        char *servIP;   // IP do servidor
        char *echoString;   // String a ser enviada ao servidor
        char echoBuffer[RCVBUFSIZE];  // String recebida de volta do servidor
        int echoStringLen;  // Tamanho da estrutura de dados do endereço do
```

```
cliente
        int bytesRcvd, totalBytesRcvd;
         WSADATA wsaData;  // Estrutura para comunicação pelo WinSock
        servIP = "192.168.0.227";
        echoString ="Teste de comunicação";
        echoServPort = 5600;
        if (WSAStartup(MAKEWORD(2, 0),&wsaData) != 0) // Carrega a winsock.dll 2.0
        {
                fprintf(stderr,"WSAStartup() failed");
                exit(1);
        }
        // Cria um socket para conexão
        if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
                DieWithError("socket() failed");
        memset(&echoServAddr, 0, sizeof(echoServAddr));
        echoServAddr.sin_family = AF_INET;
        echoServAddr.sin_addr.s_addr = inet_addr(servIP);
        echoServAddr.sin_port = htons(echoServPort);
        // Se conecta com o servidor
        if (connect(sock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr))
< 0)
                DieWithError("connect() failed");
                echoStringLen = strlen(echoString);
        // Envia para o servidor a mensagem
        if (send(sock, echoString, echoStringLen, 0) != echoStringLen)
                DieWithError("send() sent a different number of bytes than
expected");
        totalBytesRcvd = 0;
        printf("Received: ");
        while (totalBytesRcvd < echoStringLen)
        {
                // Recebe de volta a mensagem co servidor como eco
                if ((bytesRcvd = recv(sock, echoBuffer, RCVBUFSIZE - 1, 0)) <= 0)
                        DieWithError("recv() failed or connection closed
prematurely");
                totalBytesRcvd += bytesRcvd;
                echoBuffer[bytesRcvd] = '\0';
                printf(echoBuffer);
        }
        printf("\n");
        closesocket(sock);  // Fecha o socket
        WSACleanup();
        exit(0);
}
void DieWithError(char *errorMessage)
```

```
{
        fprintf(stderr,"%s: %d\n",errorMessage, WSAGetLastError());
        exit(1);
}
```

Anexo B: Programação do ambiente de simulação distribuída implantado

B.1 Biblioteca Konfort.h do Siemens Prodave 6.0

```
#ifndef KOMFORT
#define KOMFORT
#ifdef __cplusplus
   extern "C" {
#endif

/*****************************************************************************/
// error no, textbuffer

int WINAPI error_message(int,char *);

// kg value, float value
int WINAPI kg_to_float(void *,void *);
// float value, kg_value
int WINAPI float_to_kg(void *,void *);

// gp value, float value
void WINAPI gp_to_float(void *,void *);
// float value, gp value
void WINAPI float_to_gp(void *,void *);

// value, bit no
char WINAPI testbit (unsigned char,unsigned char);

// value, byte buffer
void WINAPI byte_boolean (char,char*);

// byte buffer
char WINAPI boolean_byte (char*);

// 2byte kf/kh value S5
unsigned short WINAPI kf_integer (unsigned short);

// buffer, amount bytes to swab
void WINAPI swab_buffer(void *, int);
```

```
// dest buffer, source buffer amount bytes to copy
void WINAPI copy_buffer (void *,void *, int);

// ptr value, amount values, bytechange input, bytechange output
void WINAPI USHORT_2_bcd(unsigned short *,unsigned short,char,char);

// ptr value, amount values, bytechange input, bytechange output
void WINAPI bcd_2_USHORT(unsigned short *,unsigned short,char,char);

/*********************************************************************/


#ifdef __cplusplus
    } /* extern "C" */
#endif
#endif
```

B.2 Biblioteca w95_s7.h do Siemens Prodave 6.0

```
#ifndef PRODAVE
#define PRODAVE
#ifdef __cplusplus
   extern "C" {
#endif

#define BST_IN_RAM   0x0010
#define BST_IN_EPROM 0x0020


/*********************************************************************/
/*********************************************************************/
/* typedef for addess table    max 16 connections                    */
/*********************************************************************/
#pragma pack(1)
typedef struct
        {
        unsigned char  adr;          /* stationsadresse     default 2          */
        unsigned char  segmentid;    /* segment id          default 0          */
        unsigned char  slotno;       /* slot no             default 1          */
        unsigned char  rackno;       /* rack no             default 0          */
```

```
        } adr_table_type;

typedef struct
        {
        unsigned char adr[6];        /* mac address                              */
        } mac_adr_table_type;
#pragma pack()
/*****************************************************************************/
/* connection no, device_name, connection address table */
int WINAPI load_tool(char,char *,adr_table_type *);
/* connection no, device_name, connection address table, mac address table */
int WINAPI load_tool_ex(char,char *,adr_table_type *,mac_adr_table_type *);
/*****************************************************************************/
int WINAPI unload_tool(void);
/* connection no */
int WINAPI new_ss(char);


/*****************************************************************************/
/*****************************************************************************/
// functions for AS300
/*****************************************************************************/
/*****************************************************************************/
int WINAPI ag_info(char *);
int WINAPI ag_zustand(char *);
/*****************************************************************************/
/* blockno, no, amount, buffer */
int WINAPI db_read(int,int,int*,void *);
int WINAPI db_write(int,int,int*,void *);
int WINAPI d_field_read(int,int,int,void *);
int WINAPI d_field_write(int,int,int,void *);
/*****************************************************************************/
/* no, amount, buffer */
int WINAPI e_field_read(int,int,void *);
int WINAPI a_field_read(int,int,void *);
int WINAPI m_field_read(int,int,void *);
int WINAPI t_field_read(int,int,void *);
int WINAPI z_field_read(int,int,void *);
int WINAPI a_field_write(int,int,void *);
int WINAPI m_field_write(int,int,void *);
int WINAPI z_field_write(int,int,void *);
/*****************************************************************************/
/* buffer */
int WINAPI db_buch(void *);
/*****************************************************************************/
```

```
/* data field, buffer */
int WINAPI mix_read(char*,void*);
int WINAPI mix_write(char*,void*);
/*****************************************************************************/
/* no, bitno */
int WINAPI mb_setbit(int,int);
int WINAPI mb_resetbit(int,int);
/* no, bitno, value */
int WINAPI mb_bittest(int,int,char *);
/*****************************************************************************/

/*****************************************************************************/
/*****************************************************************************/
// functions for AS200
/*****************************************************************************/
/*****************************************************************************/
// no, amount, value
int WINAPI as200_e_field_read(int,int,void*);
int WINAPI as200_a_field_read(int,int,void*);
int WINAPI as200_m_field_read(int,int,void*);
int WINAPI as200_t_field_read(int,int,void*);
int WINAPI as200_z_field_read(int,int,void*);
int WINAPI as200_sm_field_read(int,int,void*);
int WINAPI as200_vs_field_read(int,int,void*);
int WINAPI as200_a_field_write(int,int,void*);
int WINAPI as200_m_field_write(int,int,void*);
int WINAPI as200_z_field_write(int,int,void*);
int WINAPI as200_sm_field_write(int,int,void*);
int WINAPI as200_vs_field_write(int,int,void*);
// data, value
int WINAPI as200_mix_read(char*,void*);
int WINAPI as200_mix_write(char*,void*);
// no, bitno
int WINAPI as200_mb_setbit(int,int);                    /* no, bitno
*/
int WINAPI as200_mb_resetbit(int,int);                  /* no, bitno
*/
// no, bitno, value
int WINAPI as200_mb_bittest(int,int,void *);            /* no, bitno, value
*/
// value
int WINAPI as200_ag_info(void*);
int WINAPI as200_ag_zustand(void*);
/*****************************************************************************/
// functions for AS200
```

```
/*****************************************************************************/
/*****************************************************************************/
// Teleservice functions
/*****************************************************************************/
/*****************************************************************************/
#define TS_CONNECTED      0x00000001
#define TS_DISCONNECTED 0x00000003


//dial via modem, asynchronous dial if handle
//modem name, standort name, tel.no, username, password , window handel, message,
wparam,NULL
int WINAPI ts_dial(char*,char*,char*,char*,char*,HANDLE,UINT,WPARAM,char*);

//disconnect dial connection
int WINAPI ts_hang_up_dial(void);

//set indicaton on ring -> message if ts-adapter is connected
// modem name, no of rings, window handel, message, wparam,NULL
int WINAPI ts_set_ringindicator(char*,int,HANDLE,UINT,WPARAM,char*);

// information from ring connected ts-adapter
// eventid (16 bytes) , asid (1 byte)
int WINAPI ts_read_info(void *,unsigned char *);

//disconnect ring connection
int WINAPI ts_hang_up_ring(void);

// get names of all system known modems
// ModemID 0..n, ptr ModemName, ptr ModemNameBufferLen
int WINAPI ts_get_modem_name(int,char*,int*);


#ifdef __cplusplus
    } /* extern "C" */
#endif
#endif
```

B.3. Programa cliente do computador remoto

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <stdio.h>
#include <conio.h>
#include <process.h>



#include "resource.h"
#include <winsock.h>    /* for socket(),... */

#define MYTIMER 0x1234
#define MAX_MIX 4
#define MAX_BUFFER 2048
#define RCVBUFSIZE 32
#define TESTE 1000
char                            buffer2[MAX_BUFFER];
char                            buffer[MAX_BUFFER];
char    str[100];
char    str1[100];
unsigned char * buffer_byte = (unsigned char *)buffer;
unsigned char * buffer_byte2 = (unsigned char *)buffer2;
HANDLE  hInst,segment_hnd;
char    echoBuffer[RCVBUFSIZE];        /* Buffer for echo string */
int servSock;
int amount                       = 1;
int sock;
int q = 17;
int qz=0;
int q_p = 0;
char    receberServidor[8];
//char *echoString[9];
char echoString[9];
int echoStringLen;
struct sockaddr_in echoServAddr;


LRESULT CALLBACK MainWndProc( HWND, UINT, WPARAM, LPARAM );
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
BOOL WINAPI LOADBOX             ( HWND, UINT, WPARAM, LPARAM );
void ThreadProc(void *param);



/*************************************************************************/
/*         Funktion: WinMain(HANDLE, HANDLE, LPSTR, int)                 */
/*************************************************************************/
int CALLBACK WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR
lpCmdLine,int nCmdShow)
{
  MSG msg;
  if (!hPrevInstance)
     if (!InitApplication(hInstance))
        return (FALSE);

  if (!InitInstance(hInstance, nCmdShow))
     return (FALSE);

  while (GetMessage(&msg,NULL,0,0))
     {
     TranslateMessage(&msg);
     DispatchMessage(&msg);
     }
  return (msg.wParam);
} /* end of WinMain */
/*************************************************************************/


/*************************************************************************/
```

```
/*      Function: InitApplication(Handle)                              */
/*********************************************************************/
BOOL InitApplication(hInstance)
HANDLE hInstance;

{
   WNDCLASS wc;

   wc.style = 0;
   wc.lpfnWndProc = MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
   wc.hCursor = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = GetStockObject(WHITE_BRUSH);
   wc.lpszMenuName = "IDR_MENU1";
   wc.lpszClassName = "DemoWClass";

   return (RegisterClass(&wc));
} /* end of InitApplication */
/*********************************************************************/


/*********************************************************************/
/*           Function: InitInstance(HANDLE, int)                     */
/*********************************************************************/
BOOL InitInstance(hInstance, nCmdShow)
HANDLE         hInstance;
int            nCmdShow;

{
HWND        hWnd;

   hInst = hInstance;

   hWnd = CreateWindow(
          "DemoWClass",
          "Trabalho Bruno",
          WS_OVERLAPPEDWINDOW,
          100,
          100,
          200,
          100,
          NULL,
          NULL,
          hInstance,
          NULL);

   if (!hWnd) return (FALSE);

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return (TRUE);
} /* end of InitInstance */
/*********************************************************************/

/*********************************************************************/
/*             Function: MainWndProc(HWND, unsigned, WORD, LONG)     */
/*********************************************************************/
LRESULT CALLBACK MainWndProc ( HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
switch (message) {

   case WM_COMMAND:
```

```
                                    if (wParam == ID_LOAD)
                                                {
          DialogBox(hInst,MAKEINTRESOURCE(IDD_LOAD),hWnd,LOADBOX);
                                                break;
                                                }


    case WM_DESTROY:

          PostQuitMessage(0);
          break;

    default: return (DefWindowProc(hWnd, message, wParam, lParam));

    }

    return (0L);
} /* end of MainWndProc */
/************************************************************************/

/************************************************************************/
BOOL WINAPI LOADBOX(hDlg, message, wParam, lParam)
HWND hDlg;
UINT message;
WPARAM wParam;
LPARAM lParam;
{
                                HWND listbox;

                                int j;

                                unsigned short echoServPort;
                                char *servIP;

                                char echoBuffer[RCVBUFSIZE];

                                int bytesRcvd, totalBytesRcvd;
                                WSADATA wsaData;
//                              HANDLE handle;
                                int val = 0;
                                static char i;
                                struct sockaddr_in echoServAddr;


switch (message) {
    case WM_INITDIALOG:
                                 SetDlgItemText(hDlg, STATUS_SERVIDOR, "Nok");
                                 //memset(bufffer,0,MAX_BUFFER);
                                 break;
                                case WM_TIMER:
    case WM_COMMAND:
                                 if ((wParam == IDOK) || (wParam == ENVIAR) ||
((message == WM_TIMER) && (wParam == MYTIMER)))
                                        {
                                        memset(echoString,0,9);
                                        servIP = "143.107.99.178";
                                        SetDlgItemText(hDlg, IP_SERVIDOR, servIP);



                                              if(q==0){
                                                    echoString[0]='0';
                                                    echoString[1]='0';
```

```
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='1';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='1';
                //q=-1;

        }
        //if (buffer[0]== 42){
        //      q=1;
        //}

        if(q==1){
                echoString[0]='0';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='1';

        }
        if(q==2){
                echoString[0]='0';
                echoString[1]='0';
                echoString[2]='1';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='1';

        }
        if(q==3){
                echoString[0]='1';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='1';

        }
        if(q==4){
                echoString[0]='0';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='1';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='2';

        }
        if(q==5){
                echoString[0]='0';
                echoString[1]='1';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
```

```
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='2';

        }
        if(q==6){
                echoString[0]='0';
                echoString[1]='0';
                echoString[2]='1';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='2';

        }
        if(q==7){
                echoString[0]='0';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='1';
                echoString[8]='2';

        }
        if(q==8){
                echoString[0]='0';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='1';
                echoString[7]='0';
                echoString[8]='2';

        }
        if(q==9){
                echoString[0]='0';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='2';

        }
        if(q==10){
                echoString[0]='1';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='2';

        }
        if(q==11){
                echoString[0]='0';
```

```
                                        echoString[1]='0';
                                        echoString[2]='0';
                                        echoString[3]='0';
                                        echoString[4]='1';
                                        echoString[5]='0';
                                        echoString[6]='0';
                                        echoString[7]='0';
                                        echoString[8]='2';

                                }
                                if(q==12){
                                        echoString[0]='0';
                                        echoString[1]='1';
                                        echoString[2]='0';
                                        echoString[3]='0';
                                        echoString[4]='0';
                                        echoString[5]='0';
                                        echoString[6]='0';
                                        echoString[7]='0';
                                        echoString[8]='1';

                                }
                                if(q==13){
                                        echoString[0]='0';
                                        echoString[1]='0';
                                        echoString[2]='0';
                                        echoString[3]='1';
                                        echoString[4]='0';
                                        echoString[5]='0';
                                        echoString[6]='0';
                                        echoString[7]='0';
                                        echoString[8]='1';

                                }
                                if(q==14){
                                        echoString[0]='0';
                                        echoString[1]='0';
                                        echoString[2]='0';
                                        echoString[3]='0';
                                        echoString[4]='1';
                                        echoString[5]='0';
                                        echoString[6]='0';
                                        echoString[7]='0';
                                        echoString[8]='1';
                                        //q=13;

                                }
                        //      if (buffer[0]==26){
                        //              q=15;

//                              }
                                if(q==15){
                                        echoString[0]='0';
                                        echoString[1]='0';
                                        echoString[2]='0';
                                        echoString[3]='0';
                                        echoString[4]='0';
                                        echoString[5]='0';
                                        echoString[6]='0';
                                        echoString[7]='0';
                                        echoString[8]='1';

                                }
                                if(qz==9){
                                        echoString[0]='0';
                                        echoString[1]='0';
                                        echoString[2]='0';
                                        echoString[3]='0';
```

```
                echoString[4]='1';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='2';
                //q=13;
                //qz=10;

        }
        if(qz==10){
                echoString[0]='0';
                echoString[1]='1';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='1';
                //q=13;
                //qz=11;

        }
        if(qz==11){
                echoString[0]='0';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='1';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='1';
                //q=13;
                //qz=12;

        }
        if(qz==12){
                echoString[0]='1';
                echoString[1]='0';
                echoString[2]='0';
                echoString[3]='0';
                echoString[4]='0';
                echoString[5]='0';
                echoString[6]='0';
                echoString[7]='0';
                echoString[8]='2';
                q=13;
                qz=20;

        }









        qz=qz+1;
        q=q+1;
```

```
                                        echoServPort = 5600;
                                        if (WSAStartup(MAKEWORD(2, 0),&wsaData) !=
0)
                                        {
                                                SetDlgItemText(hDlg, FALHA,
"WSAStartup() failed");
                                                exit(1);
                                        }
                                        if ((sock = socket(PF_INET, SOCK_STREAM,
IPPROTO_TCP)) < 0)
                                                SetDlgItemText(hDlg, FALHA, "socket()
failed");

                                        memset(&echoServAddr, 0,
sizeof(echoServAddr));
                                        echoServAddr.sin_family = AF_INET;
                                        echoServAddr.sin_addr.s_addr =
inet_addr(servIP);
                                        echoServAddr.sin_port =
htons(echoServPort);


                                        if (connect(sock, (struct sockaddr *)
&echoServAddr, sizeof(echoServAddr)) < 0){
                                                SetDlgItemText(hDlg, FALHA,
"connect() failed");
                                                SetDlgItemText(hDlg, STATUS_SERVIDOR,
"Nok");
                                        }
                                        else
                                                SetDlgItemText(hDlg, STATUS_SERVIDOR,
"Ok");

                                        echoStringLen = strlen(echoString);


                                        if (send(sock, echoString, echoStringLen,
0) != echoStringLen)
                                                SetDlgItemText(hDlg, FALHA, "send()
sent a different number of bytes than expected");




                                        /*
                                        if (wParam == ENVIAR){
                                                handle = (HANDLE) _beginthread(
ThreadProc,0,&val); // create thread
                                                WaitForSingleObject(handle,1000);
                                        }
                                        */


                                        totalBytesRcvd = 0;

                                        /* Set the size of the in-out parameter */
                                        SetTimer(hDlg,MYTIMER,TESTE,NULL);

                                        if (wParam == FECHAR_CONEXAO)
KillTimer(hDlg,MYTIMER);
```

```
                                        while (totalBytesRcvd < echoStringLen)
                                        {
                                                if ((bytesRcvd = recv(sock,
echoBuffer, RCVBUFSIZE - 1, 0)) <= 0)
                                                        SetDlgItemText(hDlg, FALHA,
"recv() failed or connection closed prematurely");
                                                totalBytesRcvd += bytesRcvd;
                                                echoBuffer[bytesRcvd] = '\0';


                                        }

                                        buffer[0]=echoBuffer[0];
                                        buffer2[0]=echoBuffer[1];


                                        //if (receberServidor != echoBuffer)
                                        //{

                                                listbox =
GetDlgItem(hDlg,STATUS_ENTRADA);
                                SendMessage(listbox,LB_RESETCONTENT,0,0);


                                                 for (i=0;i<amount;i++)
                                                {
                                                        str1[0] = 0;
                                                        wsprintf(str, "%4d.  ",i);
                                                        j = 0x01;
                                                        while (j!=0x100)
                                                        {
                                                                if (j & buffer_byte[i])
                                                                        strcat(str1,"1");
                                                                else
                                                                        strcat(str1,"0");
                                                                j=j<<1;
                                                        }
                                                        strcat(str,str1);


                                SendMessage(listbox,LB_ADDSTRING,0,(LPARAM)str);
                                                        SetDlgItemText(hDlg,
STATUS_ENTRADA, str);
                                                }


                                            listbox =
GetDlgItem(hDlg,STATUS_SAIDA);
                                SendMessage(listbox,LB_RESETCONTENT,0,0);

                                                for (i=0;i<amount;i++)
                                                {
                                                        str1[0] = 0;
                                                        wsprintf(str, "%4d.  ",i);
                                                        j = 0x01;
                                                        while (j!=0x100)
                                                        {
                                                                if (j & buffer_byte2[i])
                                                                        strcat(str1,"1");
                                                                else
                                                                        strcat(str1,"0");
                                                                j=j<<1;
                                                        }
                                                        strcat(str,str1);
```

```
                                SendMessage(listbox,LB_ADDSTRING,0,(LPARAM)str);
                                                SetDlgItemText(hDlg,
STATUS_SAIDA, str);
                                        }


                                        UpdateWindow(hDlg);
                                //}

                        //      receberServidor[0]=echoBuffer[0];
                                //receberServidor[1]=echoBuffer[1];



                                closesocket(sock);
                                WSACleanup();
                        //} //FECHA O FOR



                        }
                        if (wParam == IDCANCEL)
                        {
                                EndDialog(hDlg, TRUE);
                                break;
                        }
                        if (wParam == FECHAR_CONEXAO)
                             {
                                //closesocket(sock);
                                //WSACleanup();
                          break;
                         }
                        if (wParam == COMECA_PRODUCAO)
                        {
                                q=0;
                                break;
                        }
                        if (wParam == PAUSA_PRODUCAO)
                        {
                                q_p=q;
                                q=18;

                                break;
                        }
                        if (wParam == REINICIA_PRODUCAO)
                        {
                                q=q_p;
                                break;
                        }
                        if (wParam == STOP_PRODUCAO)
                        {
                                q=18;
                                break;
                        }
                        if (wParam == ZERAR)
                        {
                                qz=9;
                                break;
                        }
                      default:
                      return(FALSE);
} // end of case
```

```
return (TRUE);
}  /* end of LOADBOX */
/*****************************************************************************/


void ThreadProc(void *param)
{



                                _endthread();

}
```

B.4. Programa servidor do computador local

```
/*****************************************************************************
*
*****************************************************************************/
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>

#include "resource.h"



#include <winsock.h>    /* for socket(),... */


#include "w95_s7.h"
#include "komfort.h"

#define RCVBUFSIZE 32   /* Size of receive buffer */
#define MAXPENDING 5    /* Maximum outstanding connection requests */

#define MYTIMER 0x1234
#define TESTE 500
#define MAX_MIX 4
#define MAX_BUFFER 2048
HANDLE  hInst,segment_hnd;
char                            buffer[MAX_BUFFER];
char                            buffer2[MAX_BUFFER];
char                            buffer3[MAX_BUFFER];
char    echoBuffer[RCVBUFSIZE];         /* Buffer for echo string */
char    str[100];
char    enviarCliente[8];
char    str_escrita[100];
char    str1[100];
int                              res;
int     res2;
int     res3;
int servSock;
int recvMsgSize;                    /* Size of received message */
```

```
int clntSocket;
//int servSock;                    /* Socket descriptor for server */
int clntSock;                     /* Socket descriptor for client */
struct sockaddr_in echoServAddr; /* Local address */
struct sockaddr_in echoClntAddr; /* Client address */
unsigned short echoServPort;     /* Server port */
int clntLen;            /* Length of client address data structure */



WORD *                           buffer_int = (WORD *)buffer;
unsigned char * buffer_byte = (unsigned char *)buffer;
unsigned char * buffer_byte2 = (unsigned char *)buffer2;
char    text[255];


#pragma pack(1)
adr_table_type adr_table[5] = { {2,0,2,0},
                                                                    {0,0,2,0},
                                                                    {0,0,2,0},
                                                                    {0,0,2,0},
                                                                    {0,0,0,0}};



int datatype = 'e';
int datatype2 = 'a';
int blockno                       = 10;
int no                              = 0;
int bitno                         = 0;
int amount                        = 2;
char ssno    = 1;
int i;


struct
                                  {
                                  unsigned char type;
                                  unsigned char size;
                                  unsigned short dbno;
                                  unsigned short no;
                                  } mix_tab[30];
#pragma pack()

//char sModemName[80]="Sportster 14400 FAX extern";
char sModemName[80]="Standardmodem";
char sStandortName[80]="Standardstandort";
//char sTelNo[80]="+49 (0721) 595-5587";
char sTelNo[80]="+49 (0171) 2255521";
char sUserName[80]="ADMIN";
char sPassword[80]="admin";
HWND ProgHandle;


LRESULT CALLBACK MainWndProc( HWND, UINT, WPARAM, LPARAM );
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
BOOL WINAPI LOADBOX              ( HWND, UINT, WPARAM, LPARAM );
void ThreadProc(void *param);


/*************************************************************************/
/*         Funktion: WinMain(HANDLE, HANDLE, LPSTR, int)                 */
/*************************************************************************/
int CALLBACK WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR
lpCmdLine,int nCmdShow)
{
  MSG msg;
```

```
   if (!hPrevInstance)
      if (!InitApplication(hInstance))
         return (FALSE);

   if (!InitInstance(hInstance, nCmdShow))
      return (FALSE);

   while (GetMessage(&msg,NULL,0,0))
      {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
      }
   return (msg.wParam);
} /* end of WinMain */
/************************************************************************/


/************************************************************************/
/*      Function: InitApplication(Handle)                               */
/************************************************************************/
BOOL InitApplication(hInstance)
HANDLE hInstance;

{
   WNDCLASS wc;

   wc.style = 0;
   wc.lpfnWndProc = MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
   wc.hCursor = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = GetStockObject(WHITE_BRUSH);
   wc.lpszMenuName = "IDR_MENU1";
   wc.lpszClassName = "DemoWClass";

   return (RegisterClass(&wc));
} /* end of InitApplication */
/************************************************************************/


/************************************************************************/
/*           Function: InitInstance(HANDLE, int)                        */
/************************************************************************/
BOOL InitInstance(hInstance, nCmdShow)
HANDLE        hInstance;
int           nCmdShow;

{
HWND       hWnd;

   hInst = hInstance;

   hWnd = CreateWindow(
          "DemoWClass",
          "Trabalho Bruno",
          WS_OVERLAPPEDWINDOW,
          100,
          100,
          200,
          100,

          NULL,
          NULL,
          hInstance,
          NULL);
```

```
    if (!hWnd) return (FALSE);

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return (TRUE);
} /* end of InitInstance */
/************************************************************************/



/************************************************************************/
/*             Function: MainWndProc(HWND, unsigned, WORD, LONG)        */
/************************************************************************/
LRESULT CALLBACK MainWndProc ( HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{

switch (message) {

   case WM_COMMAND:

                                  if (wParam == ID_LOAD)
                                          {
          DialogBox(hInst,MAKEINTRESOURCE(IDD_LOAD),hWnd,LOADBOX);
                                       break;
                                       }


   case WM_DESTROY:
                                 unload_tool();
        PostQuitMessage(0);
        break;

   default: return (DefWindowProc(hWnd, message, wParam, lParam));

   }

   return (0L);
} /* end of MainWndProc */
/************************************************************************/

/************************************************************************/
BOOL WINAPI LOADBOX(hDlg, message, wParam, lParam)
HWND hDlg;
UINT message;
WPARAM wParam;
LPARAM lParam;
{
BOOL result;
HWND listbox;
int show = 0;
int result2;
int j;
int loop=0;
char output;


int total=0;
static char i;
char szErrorFilePath[MAX_PATH];
char drive[MAX_PATH];
char dir[MAX_PATH];
char fname[MAX_PATH];
char ext[MAX_PATH];

WSADATA wsaData;                /* Structure for WinSock setup communication */
```

```
HANDLE handle;
int val = 0;

switch (message) {
    case WM_INITDIALOG:
                                    i = 0;
                                    //teste
                                    output = 'b';
                                    show = 1;
                                    datatype ='e';
                                    datatype2 ='a';
                                    blockno=1;
                                    //no=16;
                                    amount=1;
                                    memset( buffer,0,sizeof(buffer));


                                    //teste



                                    CheckDlgButton(hDlg, IDC_VERB1, 1);

                                    itoa(adr_table[i].adr,str,10);
                                    SetDlgItemText(hDlg, IDC_STATIONADR, str);
                                    itoa(adr_table[i].rackno,str,10);
                                    SetDlgItemText(hDlg, IDC_RACKNO, str);
                                    itoa(adr_table[i].segmentid,str,10);
                                    SetDlgItemText(hDlg, IDC_SEGMENTID, str);
                                    itoa(adr_table[i].slotno,str,10);
                                    SetDlgItemText(hDlg, IDC_SLOTNO, str);

                                    // load error text file "error.dat"
                                    GetModuleFileName(NULL,szErrorFilePath,MAX_PATH);
                                    _splitpath(szErrorFilePath,drive,dir,fname,ext);

                                   sprintf(szErrorFilePath,"%s%serror.dat",drive,dir);

                                    error_message(0,szErrorFilePath);

                                    adr_table[4].adr = 0;




                                    show = 1;
                                    break;

                                   case WM_TIMER:
   case WM_COMMAND:

                                    if ((wParam == IDC_VERB1) ||
                                          (wParam == IDC_VERB2) ||
                                          (wParam == IDC_VERB3) ||
                                          (wParam == IDC_VERB4))
                                          {
                                          show = 1;
                                          break;
                                          }




                                    if ((wParam == IDOK) ||
                                          ((message == WM_TIMER) && (wParam ==
MYTIMER)))
```

```
                                        {


                                        if (IsDlgButtonChecked(hDlg, IDC_VERB1))
ssno=1;
                                        if (IsDlgButtonChecked(hDlg, IDC_VERB2))
ssno=2;
                                        if (IsDlgButtonChecked(hDlg, IDC_VERB3))
ssno=3;
                                        if (IsDlgButtonChecked(hDlg, IDC_VERB4))
ssno=4;

                                        adr_table[i].adr       =
GetDlgItemInt(hDlg, IDC_STATIONADR, &result,0);
                                        adr_table[i].rackno    =
GetDlgItemInt(hDlg, IDC_RACKNO, &result,0);
                                        adr_table[i].slotno    =
GetDlgItemInt(hDlg, IDC_SLOTNO, &result,0);
                                        adr_table[i].segmentid =
GetDlgItemInt(hDlg, IDC_SEGMENTID, &result,0);

                                        ssno = 1;



                                        if (wParam != MYTIMER)
                                        {
                                               res = unload_tool();
                                               res =
load_tool(ssno,NULL,(adr_table_type*)adr_table);

                                        //Parte do
Sockets************************************************************

                                        echoServPort = 5600;//atoi(argv[1]);
/* first arg:  Local port */

                                        if (WSAStartup(MAKEWORD(2, 0),
&wsaData) != 0) /* Load Winsock 2.0 DLL */
                                        {
                                               fprintf(stderr, "WSAStartup()
failed");
                                               exit(1);
                                        }


                                        /* Cria o Sockets para as futuras
conecções */
                                        if ((servSock = socket(PF_INET,
SOCK_STREAM, IPPROTO_TCP)) < 0)
                                               SetDlgItemText(hDlg, FALHA,
"socket() failed");//DieWithError("socket() failed");

                                        /* Construct local address structure
*/
                                        memset(&echoServAddr, 0,
sizeof(echoServAddr));    /* Zero out structure */
                                        echoServAddr.sin_family = AF_INET;
/* Internet address family */
                                        echoServAddr.sin_addr.s_addr =
htonl(INADDR_ANY); /* Any incoming interface */
                                        echoServAddr.sin_port =
htons(echoServPort);      /* Local port */


                                        /* Bind to the local address */
                                        if (bind(servSock, (struct sockaddr
*) &echoServAddr, sizeof(echoServAddr)) < 0)
                                               SetDlgItemText(hDlg, FALHA,
```

```
"bind() failed");//DieWithError("bind() failed");

                                        /* Mark the socket so it will listen
for incoming connections */
                                        if (listen(servSock, MAXPENDING) < 0)
                                              SetDlgItemText(hDlg, FALHA,
"listen() failed");//DieWithError("listen() failed");

                                        //Final do Sockets
*****************************************************************

                                    }




                                    handle = (HANDLE) _beginthread(
ThreadProc,0,&val); // create thread
                                    WaitForSingleObject(handle,1000);


                                    SetDlgItemText(hDlg, IP_CLIENTE,
inet_ntoa(echoClntAddr.sin_addr));
                                    clntSocket=clntSock;

                                    /* Receive message from client */
                                    if ((recvMsgSize = recv(clntSocket,
echoBuffer, RCVBUFSIZE, 0)) < 0);
                                          //SetDlgItemText(hDlg, FALHA, "recv()
failed");//DieWithError("recv() failed");
                                          SetDlgItemText(hDlg,
RECEBIDO,echoBuffer );




                                    /* Set the size of the in-out parameter */
                                    SetTimer(hDlg,MYTIMER,TESTE,NULL);

                                    if (wParam == Fechar)
KillTimer(hDlg,MYTIMER);



                                    //program savety
                                    if (amount > MAX_BUFFER) amount =
MAX_BUFFER;



                                    res = new_ss((char)(i+1));
                                    res2 = new_ss((char)(i+1));

                                    if (res == 0 && res2 == 0)
                                    {

                                          /* Caso queira fazer a reação da
saída por alteração da entrada.
                                          if (buffer[0] == 1)
                                          {
                                                buffer3[0]= 15;
                                                i=0;
                                                res3 = new_ss((char)(i+1));
                                                if (res3 == 0)
```

```
                                                {
                                                        res3 =
a_field_write(no,amount,buffer3);
                                                }
                                        }
                                        */
                                        // Constroi o buffer para escrever na
saida.
                                        if (echoBuffer[0] == 49)
                                            total=total+1;
                                        if (echoBuffer[0] == 48)
                                            total=total;
                                        if (echoBuffer[1] == 49)
                                            total=total+2;
                                        if (echoBuffer[1] == 48)
                                            total=total;
                                        if (echoBuffer[2] == 49)
                                            total=total+4;
                                        if (echoBuffer[2] == 48)
                                            total=total;
                                        if (echoBuffer[3] == 49)
                                            total=total+8;
                                        if (echoBuffer[3] == 48)
                                            total=total;
                                        if (echoBuffer[4] == 49)
                                            total=total+16;
                                        if (echoBuffer[4] == 48)
                                            total=total;
                                        if (echoBuffer[5] == 49)
                                            total=total+32;
                                        if (echoBuffer[5] == 48)
                                            total=total;
                                        if (echoBuffer[6] == 49)
                                            total=total+64;
                                        if (echoBuffer[6] == 48)
                                            total=total;
                                        if (echoBuffer[7] == 49)
                                            total=total+128;
                                        if (echoBuffer[7] == 48)
                                            total=total;
                                        // no último bit enviado pelo cliente
tem o endereço do cartão onde será escrita a saída.

                                        //Vai escrever no cartão de endereço
12 se vier 0 no último bit
                                        if (echoBuffer[8] == 48)
                                            no=12;
                                        //Vai escrever no cartão de endereço
16 se vier 1 no último bit
                                        if (echoBuffer[8] == 49)
                                            no=16;
                                        //Vai escrever no cartão de endereço
20 se vier 2 no último bit
                                        if (echoBuffer[8] == 50)
                                            no=20;


                                        // Com o buffer recebido completo,
escreve ele na saida.
                                        buffer3[0]= total;
                                        i=0;
                                        res3 = new_ss((char)(i+1));
                                        if (res3 == 0)
                                            res3 =
a_field_write(no,amount,buffer3);
                                        //Ler a entrada.
                                        if (datatype == 'e')
                                            res =
e_field_read(no,amount,buffer);
```

```
                                        //Ler a saida.
                                        if (datatype2 == 'a')
                                              res2 =
a_field_read(no,amount,buffer2);

                                        enviarCliente[0]=buffer[0];
                                        enviarCliente[1]=buffer2[0];


                                    }
                                    while (recvMsgSize > 0)    // O zero indica
o fim da trasmissão pro cliente.
                                              {
                                                    //Mandando pro cliente a
leitura da saída.
                                                    if
(send(clntSocket,enviarCliente , recvMsgSize, 0) != recvMsgSize);

                                //SetDlgItemText(hDlg, FALHA, "send() failed");

                                                    //Manda pro cliente a
leitura da entrada.
                                                    //if (send(clntSocket,
buffer2, recvMsgSize, 0) != recvMsgSize);


                                                    //Verifica se tem mais
informação para receber
                                                    if ((recvMsgSize =
recv(clntSocket, echoBuffer, RCVBUFSIZE,0)) < 0);

                                //SetDlgItemText(hDlg, FALHA, "recv() failed");
                                              }

                                    closesocket(clntSocket); // Fecha a conexão
(socket) com o cliente.


                                    if (res != 0 || res2 != 0 || res3 != 0)
                                          {
                                          KillTimer(hDlg,MYTIMER);
                                          EndDialog(hDlg, TRUE);
                                          return (TRUE);
                                          }


                                    //if ((message == WM_TIMER) && (wParam ==
MYTIMER))
                                          //SetTimer(hDlg,MYTIMER,TESTE,NULL);


                                    show = 1;
                                    break;


                                }


                                if (wParam == Fechar) KillTimer(hDlg,MYTIMER);

                                if (wParam == IDCANCEL)
                                      {
                                      KillTimer(hDlg,MYTIMER);
                                      EndDialog(hDlg, TRUE);
                                      show = 1;
                                      break;
                                      }
```

```
                                if (wParam == Fechar)
                                    {
                                    res = unload_tool();
                                  //closesocket(clntSocket); // Fecha a conexão
(socket) com o cliente.
                                  show=1;
                                  break;
                                  }


                               default:
                                 return(FALSE);
  } // end of case


if (show == 1)
                               {
                               adr_table[i].adr       = GetDlgItemInt(hDlg,
IDC_STATIONADR, &result,0);
                               adr_table[i].rackno    = GetDlgItemInt(hDlg,
IDC_RACKNO, &result,0);
                               adr_table[i].slotno    = GetDlgItemInt(hDlg,
IDC_SLOTNO, &result,0);
                               adr_table[i].segmentid = GetDlgItemInt(hDlg,
IDC_SEGMENTID, &result,0);

                               if (IsDlgButtonChecked(hDlg, IDC_VERB1)) i=0;
                               if (IsDlgButtonChecked(hDlg, IDC_VERB2)) i=1;
                               if (IsDlgButtonChecked(hDlg, IDC_VERB3)) i=2;
                               if (IsDlgButtonChecked(hDlg, IDC_VERB4)) i=3;

                               itoa(adr_table[i].adr,str,10);
    SetDlgItemText(hDlg, IDC_STATIONADR, str);
                               itoa(adr_table[i].rackno,str,10);
    SetDlgItemText(hDlg, IDC_RACKNO, str);
                               itoa(adr_table[i].segmentid,str,10);
    SetDlgItemText(hDlg, IDC_SEGMENTID, str);
                               itoa(adr_table[i].slotno,str,10);
    SetDlgItemText(hDlg, IDC_SLOTNO, str);


                               result2 = new_ss((char)(i+1));

                               if (result2 == 0)
                                 SetDlgItemText(hDlg, status_conexao, "ok");
                               if (result2 != 0)
                                 SetDlgItemText(hDlg, status_conexao, "Nok");


                               listbox = GetDlgItem(hDlg,mostra_entrada);
                               SendMessage(listbox,LB_RESETCONTENT,0,0);


                                   for (i=0;i<amount;i++)
                                       {
                                       str1[0] = 0;
                                       wsprintf(str, "%4d.  ",i);
                                       j = 0x01;
                                       while (j!=0x100)
                                             {
                                             if (j & buffer_byte[i])
                                                   strcat(str1,"1");
                                             else
                                                   strcat(str1,"0");
                                             j=j<<1;
                                             }
                                       strcat(str,str1);
```

```
                                SendMessage(listbox,LB_ADDSTRING,0,(LPARAM)str);
                                        SetDlgItemText(hDlg, mostra_entrada, str);
                                        }




    listbox = GetDlgItem(hDlg,mostra_saida);
                                SendMessage(listbox,LB_RESETCONTENT,0,0);

                                    for (i=0;i<amount;i++)
                                        {
                                        str1[0] = 0;
                                        wsprintf(str, "%4d.  ",i);
                                        j = 0x01;
                                        while (j!=0x100)
                                                {
                                                if (j & buffer_byte2[i])
                                                        strcat(str1,"1");
                                                else
                                                        strcat(str1,"0");
                                                j=j<<1;
                                                }
                                        strcat(str,str1);

                                SendMessage(listbox,LB_ADDSTRING,0,(LPARAM)str);
                                        SetDlgItemText(hDlg, mostra_saida, str);
                                        }


                                UpdateWindow(hDlg);
                                }

return (TRUE);
}  /* end of LOADBOX */
/*************************************************************************/

unsigned long hexstr2value(char * s)
{
return(strtol(s,NULL,16));
}
unsigned long decstr2value(char * s)
{
return(atol(s));
}
unsigned long binstr2value(char * s)
{
unsigned long v = 0;
unsigned long w = 1;
int i,j;

j = strlen(s);
for (i=0;i<j;i++)
                                {
                                if (s[i] == '1') v = v | w;
                                w = w << 1;
                                }
return(v);
}
```

```
void ThreadProc(void *param)
{

                                //int h=*((int*)param);
                                //printf("%d Thread is Running!\n",h);
                                clntLen = sizeof(echoClntAddr);
                                                /* Wait for a client to
connect */
                                            if ((clntSock = accept(servSock,
(struct sockaddr *) &echoClntAddr, &clntLen)) < 0)
                                                //SetDlgItemText(hDlg, FALHA,
"accept() failed");//DieWithError("accept() failed");



                                _endthread();

}
```