FERNANDA MARTINS MARQUES E RODRIGO ARB DE CASTRO

COMANDO E MONITORAMENTO REMOTO DE SISTEMAS PRODUTIVOS

São Paulo
2007

FERNANDA MARTINS MARQUES E RODRIGO ARB DE CASTRO

COMANDO E MONITORAMENTO REMOTO DE SISTEMAS PRODUTIVOS

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro

Área de concentração:
Engenharia Mecatrônica

Orientador: Prof. Dr. Paulo Eigi Miyagi

São Paulo
2007

FICHA CATALOGRÁFICA

Marques, Fernanda Martins
Comando e monitoramento remoto de sistemas produtivos / F.M. Marques, R.A. de Castro. -- São Paulo, 2007.
67 p.

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

1.Controladores programáveis (Monitoramento) 2.Telecomunicações I.Castro, Rodrigo Arb de II.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos III.t.

Resumo

A globalização dos mercados, a tendência de distribuição geográfica das plantas industriais e a evolução de tecnologias de telecomunicações e de mecatrônica têm motivado a o desenvolvimento de novos conceitos e técnicas para a tele-operação de sistemas produtivos. Neste contexto, existe especial interesse na implementação de sistemas de remotos de gerenciamento que incluem a monitoração, comando e integração de diferentes controladores via Internet. Assim, este trabalho apresenta o estudo e a implementação de um sistema de comando e monitoração de estações de trabalho via Internet. O estudo de caso considerado envolve duas estações de trabalho que fazem parte de um sistema de manufatura que executa a montagem automática de variações de um mesmo produto

Palavras-chave*:* Telecomando, Monitoração remota, Sistemas Produtivos, Controlador programável.

Abstract

The globalization and the trend for geographic distribution of production plants, and the use in large of telecommunication and mechatronics technologies motivates the development of new concepts and techniques for teleoperation of productive systems. In this context, there is interest in the implementation of remote management systems, which includes the monitoring, command and integration of different control units via Internet. Then, this work presents the study and implementation of command and monitoring of workstations via Internet. The case study is workstations that compose a manufacturing system, which assembly automatically variations of a basic product.

Keywords: Telecommand, Remote monitoring, Production systems, Programmable control.

SUMÁRIO

1. Introdução

Na evolução dos processos produtivos são identificados marcos de fundamental importância a partir dos quais pôde-se notar alterações definitivas no panorama industrial mundial (Lee; Lau, 1999), (Shi; Gregory, 1998). Entre eles destacam-se o surgimento das linhas de produção (sistematização e otimização de um processo especializado de manufatura) em meados do século XVIII, a incorporação de sistemas computacionais para automação de processos industriais na década de 70 e, mais recentemente, a integração dos diversos equipamentos e computadores de uma indústria em um único sistema, eliminando o conceito de ilhas de automação. Com a globalização, um maior número de indústrias de manufatura tem se estabelecido de forma distribuída, onde não apenas componentes de produtos são produzidos em diferentes plantas, mas em diferentes países, sendo então reunidos para a montagem dos produtos finais. A adoção desta estrutura distribuída envolve uma nova tecnologia: o monitoramento e o comando à distância de sistemas industriais (Martin, 1995).

Neste contexto se insere o projeto Telecomando e Monitoramento Remoto de Sistemas de Manufatura, em desenvolvimento no grupo da Mecatrônica da EPUSP, com apoio da FAPESP. Este projeto é parte do Programa TIDIA (Tecnologia da Informação no Desenvolvimento da Internet Avançada), na modalidade Kyatera, que visa o estabelecimento, de forma cooperativa, de uma rede de comunicação óptica de alto desempenho interligando laboratórios de pesquisa para o estudo, desenvolvimento e demonstração de tecnologias e aplicações da Internet Avançada. Entre os objetivos deste Projeto FAPESP, destaca-se o estudo e desenvolvimento de sistemas de gerenciamento remoto para sistemas integrados de manufatura, isto é, projeto e implementação do monitoramento e comando via internet das estações de trabalho de um sistema de manufatura.

Neste sentido, o objetivo específico deste trabalho é o desenvolvimento de um programa computacional que permita o acionamento via internet das estações de distribuição e de testes do sistema flexível de manufatura existente e em operação na EPUSP.

.

2. Fundamentos Considerados

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

Do ponto de vista da teoria, foram considerados inicialmente os **diagramas da UML** (*Unified Modelling Language*) para elaborar a especificação do programa a ser desenvolvido, contudo, na seqüência do projeto, modelagens em **Redes de Petri** complementaram esta atividade. Foram também objeto de estudos os conceitos de **controladores programáveis (CPs)**, da **orientação a objetos,** da **comunicação entre controladores programáveis e computadores** e da comunicação entre computadores pela *Internet*, via ***Sockets*** para o desenvolvimento do projeto. No que se refere à implementação, a linguagem de programação de CP adotada foi o **STL**, e a linguagem de programação de software adotada foi o **C++**. Entre as ferramentas utilizadas, destacam-se o *software* de programação dos CPs da empresa Festo (FST) e o ambiente de programação Borland C++ Builder versão 6.0. Foi ainda utilizado o novo componente do C++ Builder, que permite a visualização das imagens de uma *webcam* na interface de um programa e que foi implementado para realizar o **monitoramento visual do processo**. Foram ainda estudados os novos conceitos de *sockets* necessários para realizar a transferência de vídeo pela *Internet*.

2.1. UML – *Unified Modeling Language*

Como o próprio nome já indica, a UML (*Unified Modeling Language*) é uma linguagem de modelagem. Ela estabelece uma notação para capturar e comunicar a estrutura e o comportamento de um sistema orientado a objetos. De uma forma geral, pode-se descrever a UML como uma linguagem que, a partir de alguns elementos básicos e dos tipos de relação entre estes elementos, permite a descrição de diferentes aspectos estruturais e comportamentais do sistema através da construção de uma série de diagramas (Bueno, 2002).

2.1.1. Diagramas

Os diagramas UML permitem visualizar sistemas orientados a objetos de uma maneira funcional, permitindo uma melhor visualização de como controlá-lo e organizá-lo. A seguir apresentam-se as definições dos diagramas mais relevantes considerados para o presente projeto:

Diagrama de Casos de Uso: os diagramas de casos de uso mostram, de forma estática, as possíveis interações entre o sistema e elementos externos, os chamados *atores*. Ele apresenta uma visão ampla e genérica da funcionalidade do sistema, de uma forma que é facilmente

compreensível por pessoas de formação não necessariamente técnica, como gerentes, usuários, clientes, etc.

***Diagrama de Classes*:** este tipo de diagrama indica um conjunto de *classes* que fazem parte do sistema e as relações entre estas *classes*. Cada tipo de relação é representado por um arco diferente. Além do tipo de relação é possível especificar a multiplicidade, isto é a quantidade de *objetos* envolvidos em cada relação.

***Diagrama de Atividades*:** o diagrama de atividade combina características derivadas de redes de Petri para permitir a especificação de comportamentos como paralelismo e sincronismo. Ele descreve as seqüências de atividades realizadas, por exemplo, para um cenário.

2.2. Redes de Petri

A rede de Petri é uma técnica matemática e gráfica que oferece um ambiente uniforme para a modelagem, análise e projeto de sistemas a eventos discretos.

Como técnica matemática, um modelo em redes de Petri pode ser descrito por um sistema de equações lineares, que refletem o comportamento dinâmico do sistema, o que possibilita a análise do mesmo através da verificação formal de suas propriedades (Cassandras; Lafortune, 1999).

2.2.1. Elementos estruturais básicos da rede de Petri

Transições: correspondem a um evento que causa a mudança de estado do sistema. Podem ser temporizadas.

Lugares: representam condições (pré e pós-condições) que podem estar associadas ao modo de operação ou à disponibilidade de um recurso no sistema.

Marcas: indicam a manutenção de condições. A existência de uma *marca* num lugar eqüivale ao valor binário "1" e é "0" quando não existe *marca* no lugar.

Arcos orientados: estabelecem relações causais entre os eventos e as condições e vice-versa.

Portas: que habilitam ou inibem a ocorrência dos eventos correspondentes às *transições*, sendo denominadas *habilitadoras* ou *inibidoras*, conforme sua natureza. Para a *porta habilitadora*, quando o sinal de origem for equivalente ao valor binário "1"; esta *porta* habilita a *transição* à qual está conectada. Para a *porta inibidora,* quando o sinal de origem for equivalente ao valor binário "1", esta *porta* inibe a *transição* em que está conectada.

A Figura 1 mostra uma rede de Petri com seus principais componentes.

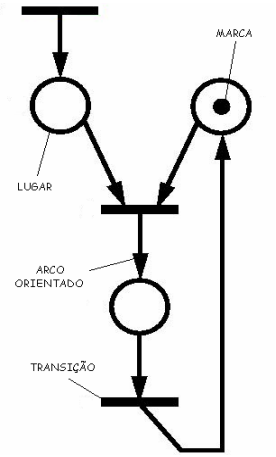


Figura 1 – Exemplo de uma rede de Petri

2.3. Controlador Programável (CP)

O controlador lógico programável – CLP – nasceu dentro da empresa General Motors, em 1968, devido à grande dificuldade de alterar a lógica de controle dos painéis de comando a cada mudança na linha de montagem. Tais mudanças implicavam em altos gastos de tempo e dinheiro (Souza, 2001).

As especificações iniciais requeriam um sistema com a flexibilidade do computador, capaz de suportar o ambiente industrial, isto é, ser facilmente programado e reprogramado, com manutenção fácil e ser também facilmente expansível e utilizável.

Devido ao intuito inicial de substituírem os painéis de relés no controle discreto, foram chamados de controladores lógicos programáveis - CLP (*Programmable Logic Controllers - PLC*). Porém, atualmente, os controladores dispõem de recursos para realizarem funções relativamente complexas não se limitando a lógica do tipo E/OU, motivo pelo qual passaram a ser chamados apenas de controladores programáveis – CP (Souza, 2001).

2.3.1. Componentes Básicos

O controlador programável é, segundo a NEMA - *National Electrical Manufacturers Association*, "um aparelho eletrônico digital que usa uma memória programável para o armazenamento interno de instruções a fim de implementar funções específicas tais como lógica, seqüenciamento, temporização, contagem e operações aritméticas, para controlar máquinas ou processos através de módulos de entradas/saídas de sinais analógicos ou digitais". O diagrama de blocos do controlador é mostrado na Figura 2:

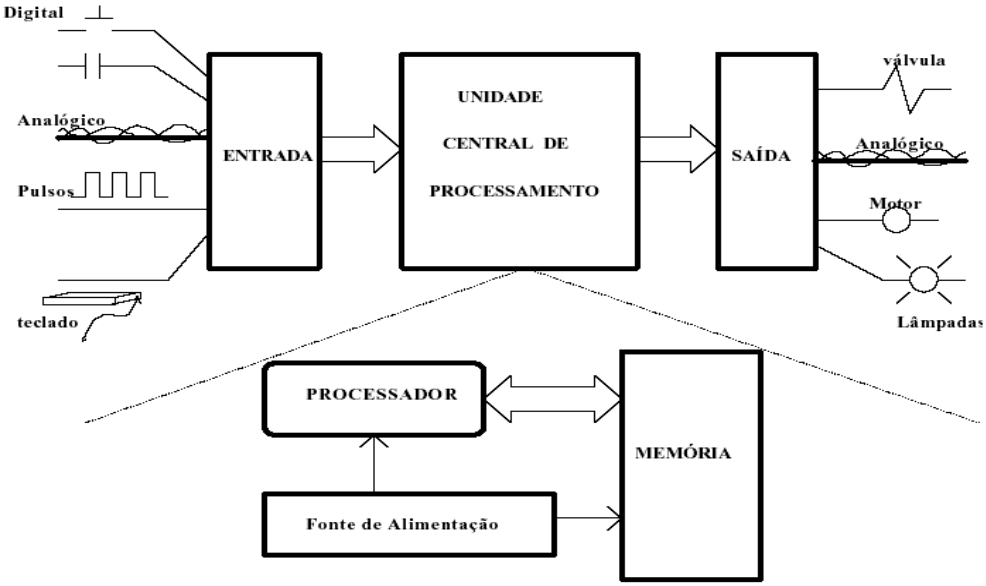


Figura 2 – Componentes Básicos do CP

Um controlador programável é formado por cinco elementos básicos: processador, memória, sistema de entradas/saídas, fonte de alimentação e terminal de programação.

As três partes principais (processador, memória e fonte de alimentação) formam o que chamamos de CPU - unidade central de processamento. O processador lê dados de entrada de vários dispositivos, executa o programa do usuário armazenado na memória e envia dados de saída para comandar os dispositivos de controle. Este processo de leitura das entradas, execução do programa e controle das saídas é feito de uma forma cíclica que é chamado de ciclo de varredura. O sistema de entrada/saída de sinais forma a interface pela qual os dispositivos de campo são conectados ao controlador. O propósito desta interface é condicionar os vários sinais recebidos ou enviados ao mundo externo. Sinais provenientes de dispositivos de comando e detecção tais como *push-buttons*, chaves limites, sensores analógicos, chaves seletoras e chaves tipo tambor (*thumbwheel*), são conectados aos terminais dos módulos de entrada. Dispositivos de atuação e de monitoração, como válvulas solenóides, lâmpadas pilotos e outros, sãos conectados aos terminais dos módulos de saída. A fonte de alimentação fornece a energia necessária para a devida operação do CP e da interface dos módulos de entrada e saída.

Outro componente do controlador programável é o dispositivo de programação. Embora seja considerado como parte do controlador, o terminal de programação, como antes era chamado, é requerido apenas para carregar o programa de aplicação na memória do controlador. Uma vez carregado o programa, o terminal pode ser desconectado do

controlador. Atualmente utiliza-se um microcomputador para programar o CP e devido à capacidade de processamento do mesmo, este também é utilizado na edição e depuração do programa.

2.3.2. Programação de CP

Existem cinco tipos básicos de linguagem de programação que normalmente são encontrados em CP padronizados pela norma IEC 61131-3: linguagens de relés ou diagrama de contatos; linguagens por blocos funcionais; SFC - *Sequential Function Chart* (fluxogramas funcionais); lista de instruções; texto estruturado (Warnock, 1988).

Dos tipos apresentados acima, o mais difundido e encontrado em quase todos os controladores é a linguagem de relés. Os blocos funcionais também podem ser encontrados com facilidade, sendo este último uma extensão do primeiro no sentido de incluir instruções mais poderosas. Atualmente o SFC, derivado das Redes de Petri, vem recebendo várias implementações, firmando-se como uma linguagem própria para processos de automação. Os fluxogramas funcionais apresentam uma variação voltada para a implementação física.

No Anexo A está a descrição de como na prática se faz a programação dos CP da Festo.

2.4. Orientação a Objetos

A orientação a objetos surgiu na década de 60 no âmbito da Engenharia de Software como uma nova forma de programação. Até então, os programas computacionais eram definidos como sendo compostos por uma coleção de funções que por sua vez podiam ser decompostas em funções mais primitivas. O enfoque principal era na seqüência de operações realizadas pelo programa. Esta forma de programação é conhecida como programação estruturada. Em contrapartida, na programação orientada a objetos o programa é composto por uma coleção de 'objetos' que interagem entre si para executar determinadas operações (Bueno, 2002). Estes objetos são formados por uma associação de dados (atributos) e funções (operações) (Figura 3).

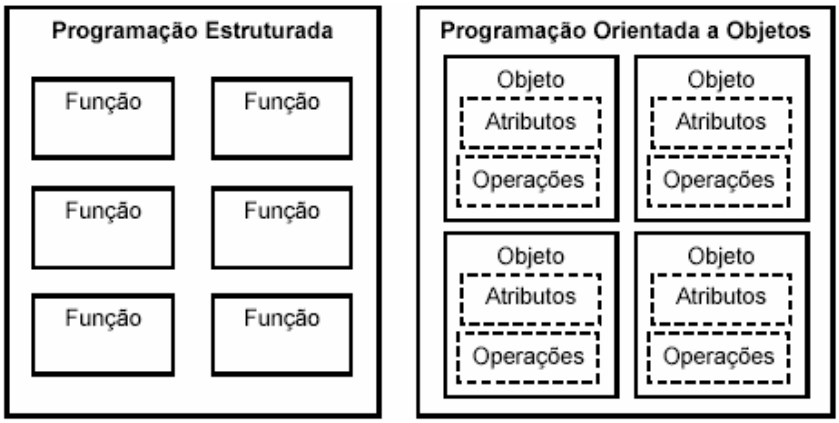


Figura 3 - Programação Estruturada x Programação Orientada a Objetos.

2.4.1. Conceitos

A orientação a objetos tem uma série de conceitos que auxiliam a delinear claramente o problema e a identificar os *objetos* e seus relacionamentos. Descrevem-se a seguir os conceitos básicos da análise orientada a objeto, isto é, a *abstração*, o *objeto*, as *classes*, os *atributos* e os *métodos*.

Abstração: genericamente, abstração significa considerar isoladamente coisas que estão unidas, ou seja, partindo do enfoque global de um determinado problema, procura-se separar os elementos fundamentais e colocá-los de uma forma mais próxima da solução. A idéia da abstração é identificar os elementos essenciais de um problema e suas propriedades fundamentais, separando ocorrências e atributos acidentais. Para a análise orientada a objetos, *abstração* é o processo de identificação dos *objetos* e seus relacionamentos. A análise orientada a objetos permite ao programador concentrar-se no que um *objeto* é e o que ele faz sem se preocupar em *como* ele o faz. A *abstração* se dá em diferentes níveis: inicialmente abstrai-se o *objeto*; de um conjunto de *objetos* cria-se um conjunto de *classes* relacionadas, de um conjunto de *classes* cria-se uma biblioteca de *classes*.

Objeto: Um *objeto* tem determinadas propriedades que o caracterizam, e que são armazenadas no próprio *objeto*. As propriedades de um *objeto* são chamadas ainda de *atributos*. O *objeto* interage com o meio e em função de excitações que sofre, realiza determinadas ações que alteram o seu estado (seus atributos). Os *atributos* de um *objeto* não são estáticos, ou seja, eles sofrem alterações com o tempo. Do ponto de vista da programação orientada a objetos, um *objeto* é uma entidade única que reúne *atributos* e *métodos*, ou seja, reúne as propriedades do *objeto* e as reações às excitações que sofre (AP, 2007). Quando se tem uma instância de uma *classe*, tem-se um *objeto* desta *classe*.

Classe: uma *classe* descreve um grupo de *objetos* com os mesmos *atributos* e comportamentos, além dos mesmos relacionamentos com outros *objetos*. Uma *classe* pode ser vista como um conjunto de códigos de programação que incluem a definição dos *atributos* e dos *métodos* necessários para a criação de um ou mais *objetos* (AP, 2007).

Método: para qualquer *objeto* pode-se relacionar determinados comportamentos, ações e reações. As ações ou comportamentos dos *objetos* são chamados de *métodos*. Assim, um *método* é uma função, um serviço fornecido pelo *objeto*. Os comportamentos dos *objetos* são definidos na *classe* através dos *métodos* e servem para manipular e alterar os *atributos* do *objeto* (alteram o estado do *objeto*).

2.5. Linguagem de Programação para Computadores: o C++

A linguagem C++ é considerada própria para implementar funcionalidades que envolvem formulações altamente abstratas como "classes", permitindo um trabalho de alto nível (trabalha-se ao nível de conceitos) e formulações de baixo nível, como o uso de chamadas de interrupções que realizam tarefas altamente específicas no nível de hardware do processador (Votre, 1998). Segundo os pesquisadores, o C++ proporciona ainda elevada produtividade, grande reaproveitamento de código, além de facilidade de extensão e manutenção. Os seus principais conceitos serão abordados a seguir.

2.5.1. Conceitos Básicos

Alguns dos conceitos básicos relacionados à programação em C++ são apresentados a seguir (Barros, 2002).

Arquivo: é um texto contendo código fonte em C++ e comandos para o pré-processador.

Comentários: um comentário em C++ usa duas barras (//).

Exemplo:

```
Aqui é programa ; //Aqui é comentário.
//Todo o resto da linha passa a ser um comentário.
```

Identificadores: seqüência de letras definidas pelo programador (nome dos *objetos*, nome dos *atributos* e *métodos*).

Palavras Chaves: são de uso interno do C++, têm significado para a linguagem, para o processo de compilação. Seus significados não podem ser alterados pelo usuário.

Operadores: símbolos cuja utilidade já é definida pelo C++; os operadores de C++ são:

```
! % _& * () - + = { } [] n ; ' : " < > ? , . /.
```

Nome: um nome denota um *objeto*, uma função, um enumerador, um tipo, um membro de *classe*, um modelo, um valor ou um *label*.

Atribuição: quando se armazena algum valor no *objeto*.

Declaração: diz que existe um *objeto* com nome qualquer, mas não cria o *objeto*. Uma declaração pode ser repetida.

Definição: cria um ou mais *objetos* e reserva memória. Uma definição não pode ser repetida.

Escopo: define onde um *objeto* é visível. Pode ser um *objeto* local, de função, de arquivo, de *classe* ou global.

Blocos: um bloco inicia com um "{" e termina com um "}". *Objetos* criados dentro do bloco são automaticamente destruídos quando o bloco é encerrado. *Objetos* criados dentro do bloco não podem ser acessados externamente (escopo).

Exemplo:

```
int main()
{ //inicio do bloco
} //fim do bloco
```

Diretrizes de pré-processamento: são informações/instruções que são passadas para o compilador com o símbolo #.

A seguir são apresentados alguns exemplos de declaração e definição de classes assim como de instanciação de *objetos*.

Exemplo 1)

Classe:

```
class TNome
{
//Atributos
tipo nome;
static tipo nome;
const tipo nome;
mutable tipo nome;
volatile tipo nome;
//Métodos
tipo função(parâmetros);
tipo função(parâmetros) const ;
```

```
static tipo função(parâmetros);
inline tipo função(parâmetros);
virtual tipo função(parâmetros);
virtual tipo função(parâmetros)=0;
};
```

Acima tem-se um modelo padrão de declaração e definição de uma *classe*. Importante: observe a presença de (;) no final do bloco que declara a *classe*.

Exemplo 2)

Objeto:

```
Tnome nomeobjeto;
```

Declara-se um objeto de uma determinada *classe* colocando o tipo do *objeto* (a *classe* a qual pertence) seguido do nome do *objeto* criado. A partir de então, manipula-se o *objeto* da maneira desejada de acordo com seus atributos e métodos (definidas na classe) (Pohl, 1991).

Exemplo 3)

```
//-----------------------------------TEndereco.h
#include <string>
class TEndereco
{
//---------------------------------Atributo
int numero;
string rua;
//---------------------------------Métodos
int Getnumero();
string Getrua();
};
//-----------------------criação de um objeto
int main()
{
```

TEndereco rodrigo;

}

Criou-se a partir da *classe* Tendereco um *objeto* que indica uma rua e um número relativo a um endereço (*atributos*) e que poderão ser alterados (através dos *métodos* da *classe*).

2.5.2. Ambiente de Programação e o Compilador

O compilador considerado no presente trabalho é o Borland C++ Builder versão 6.0. Este compilador possui interface com usuário composto pela janela 'form' na qual é desenvolvida a interface do programa a se construir, pela janela 'unit' onde é escrito o código fonte do programa, pela 'paleta de componentes' com os recursos (visuais ou não) que se pode utilizar na confecção do programa, pelo 'inspetor de objetos' que auxilia na manipulação dos recursos (*objetos*) utilizados na criação do programa e pela 'árvore de objetos' que permite uma visualização geral dos *objetos* e respectivos agrupamentos (*classes*) (Almeda, 2003).

A Figura 4, abaixo, ilustra a tela do ambiente de programação:

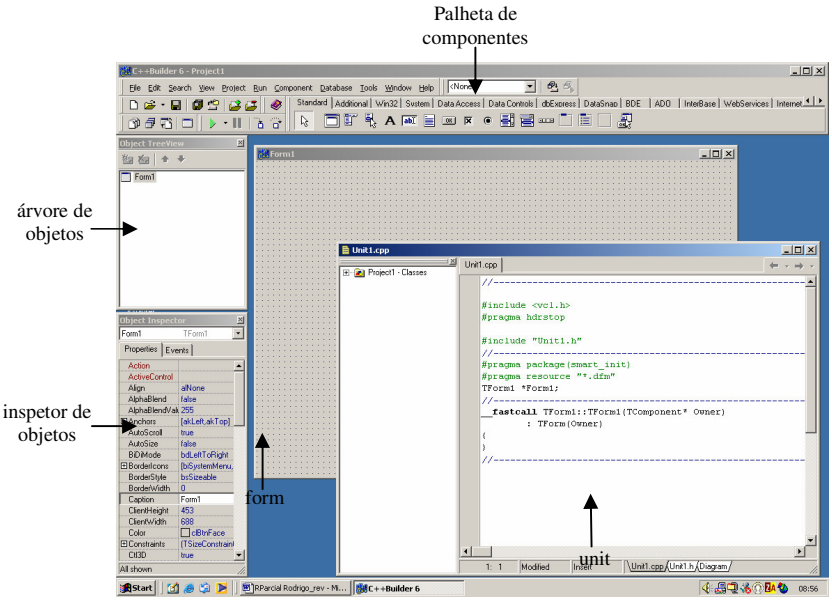


Figura 4 – Interface principal do ambiente de programação.

O compilador utilizado possui duas vantagens; a primeira consiste numa interface com o usuário relativamente simples, intuitiva e de fácil manipulação; a outra vantagem está no fato de o código fonte básico ser automaticamente fornecido após adicionar-se cada elemento à janela 'form' (DBCB, 2007).

2.5.3. Sockets

Em se tratando de hardware, os *sockets* são como encaixes para o processador, módulos de memória e outros componentes do computador. Mas, no contexto de software, os *sockets* são módulos de programas que conectam os aplicativos ao protocolo da rede de comunicação, facilitando o trabalho do programador, que precisa apenas instruir o programa a abrir um dos *sockets* disponíveis. O programa passa então a enviar e receber dados através da rede de comunicação (GDH, 2007). A Figura 5 mostra uma representação esquemática da utilização de sockets:

Figura 5 – Esquema representando a comunicação entre servidor-cliente via sockets.

No caso de C++, e, mais especificamente, no caso do compilador Borland C++ Builder 6.0, a programação relativa a sockets consiste na adição de um *objeto* ClientSocket (no caso de um programa cliente) ou ServerSocket (no caso de um programa servidor) à interface (este é um *objeto* não-visual), na definição de seus principais *atributos* e, finalmente, 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:

Host: geralmente utiliza-se o IP do servidor a se conectar.
Port: determinação da porta onde será efetuada a conexão com a Internet via *sockets*.

Os principais *métodos* do *objeto* cliente são:

ClientSocketConnect: atividades realizadas pelo cliente quando este se conecta com o servidor.
ClientSocketDisconnect: atividades realizadas pelo cliente quando este se desconecta do servidor.

ClientSocketRead: atividades realizadas pelo cliente quando o servidor manda informações para o mesmo.
ClienteSocketError: atividades realizadas pelo cliente quando este não consegue estabelecer conexão com o servidor.

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

Os principais *métodos* do *objeto* servidor são:
ServerSocketClientConnect: atividades realizadas pelo servidor quando quando o cliente se conecta.
ServerSocketClientDisconnect: atividades realizadas pelo servidor quando o cliente se desconecta.
SeverSocketClientRead: atividades realizadas pelo servidor quando o cliente manda informações.
ServerSocketAccept: atividades realizadas pelo servidor quando este aceita conexão com o cliente.

2.6. Comunicação entre CP e computador

As principais opções para a implementação da comunicação entre os CP e o computador local consideradas no presente trabalho foram: rede PROFIBUS; Ethernet; comunicação serial.

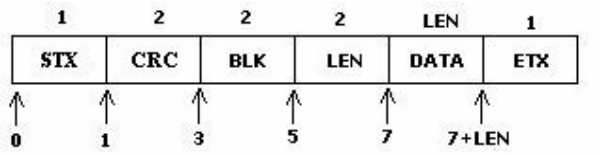
No caso da rede PROFIBUS, a comunicação envolve recursos que estão disponíveis somente em certos CPs de fabricantes que adotam o PROFIBUS.

Por outro lado, uma rede Ethernet não se justifica quando compara-se o custo de soluções com comunicação serial.

Desta forma, a Comunicação Serial mostrou-se o mais conveniente para o caso de estudo considerado e ela é, portanto, descrita a seguir.

2.6.1. Comunicação serial

Na comunicação serial, apenas um bit é enviado de cada vez através da porta de comunicação serial (do CP ou do computador). A comunicação utiliza-se de uma linguagem de programação para a confecção dos protocolos responsáveis pelo envio e recebimento de informações. Os dados enviados e recebidos são encapsulados em blocos, ou pacotes, cujo formato é o seguinte:



- STX (0x2): *byte* que representa o início de um pacote;
- CRC: contador de referência cíclico, usado para verificação da integridade do pacote;
- BLK: número do pacote (16 bits);
- LEN: número de *bytes* de dados contidos no pacote (16 bits);
- DATA: *bytes* de dados do pacote de tamanho LEN ⇐ MAX_BLOCK_LEN
- MAX_BLOCK_LEN: é uma constante (por ex.: 512);
- ETX (0x3): *byte* que representa o fim de um pacote.

O último *byte* de cada pacote deve necessariamente ser NULL, indicando o fim da informação transmitida.

2.6.2. Comunicação serial em C++

Em C++ existem *classes* já elaboradas e disponibilizadas por diversos autores que permitem a realização da comunicação serial. Esta seção apresenta uma descrição destas classes (Traverse, 2007).

Para que seja possível a comunicação serial, é preciso estabelecer uma série de definições, tais como a porta a ser utilizada (COM1 ou COM2 no caso deste projeto), tempos e constantes, velocidade e paridade (parâmetros da comunicação). Definidos todos os parâmetros, é necessário criar rotinas para recepção e envio de dados, além, é claro, do tratamento de erros no estabelecimento da comunicação, na abertura e no fechamento das portas.

Com todas estas rotinas implementadas, é possível estabelecer, de forma relativamente simples, a comunicação serial através da linguagem C++.

2.7. Monitoração Visual

2.7.1. Visualização de imagens

O componente VIDEO CAPTURE para C++ Builder da empresa XtimSoft (XTIMSOFT, 2007) consiste em uma "janela" a ser adicionada na interface de um programa qualquer. O acionamento desta "janela" a partir de um determinado evento faz com que as imagens captadas por uma *webcam* instalada no computador sejam apresentadas na interface deste programa através desta "janela". A Figura 6 ilustra qual o procedimento para adicionar o componente à interface de um programa.

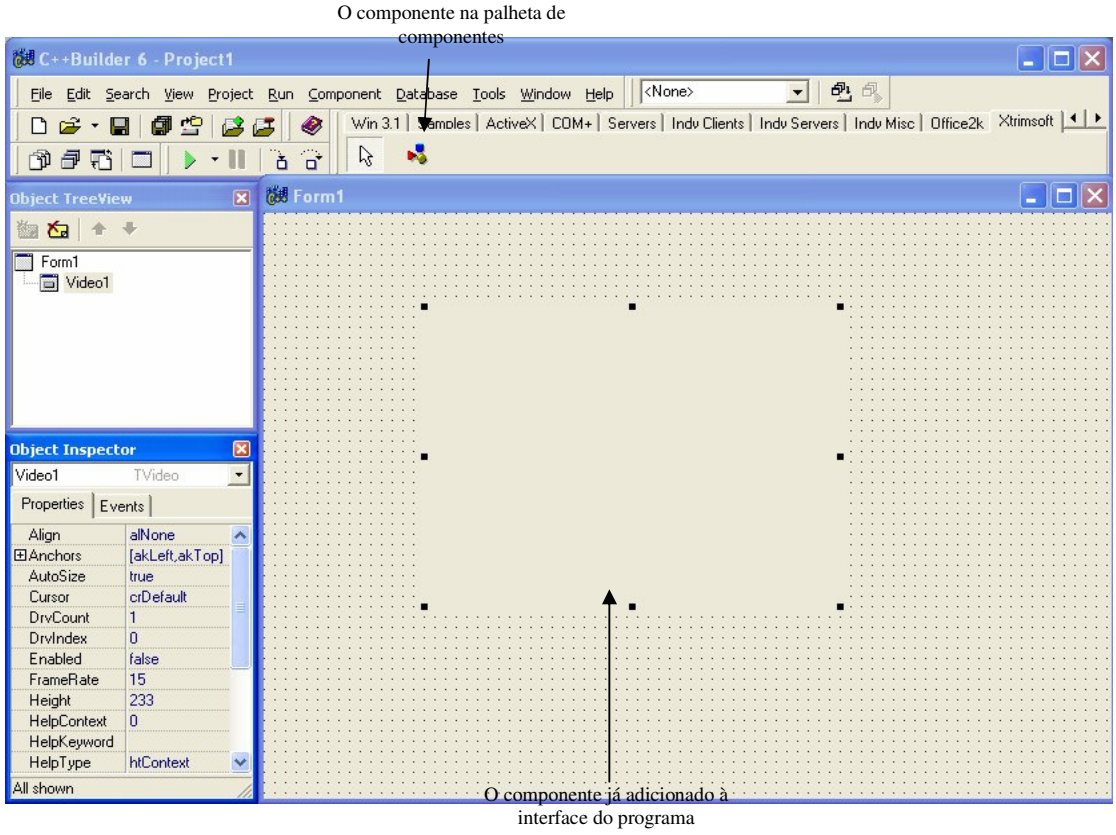


Figura 6 – Utilizando o componente VIDEO CAPTURE.

2.7.2. Transmissão de imagens pela Internet

Utilizando o componente VIDEO CAPTURE num computador com conexão à Internet e um componente de imagem (IMAGE) num outro computador também conectado à Internet pode-se estabelecer uma transmissão de imagens.

A *webcam* instalada num computador captura um *frame* que é salvo em um arquivo BITMAP, em seguida, o arquivo é transformado em uma *stream* para que esta seja transmitida pela Internet. Ao receber uma *stream* o outro computador já sabe se tratar de uma imagem e exibe-a em seu componente de imagem.

A sucessão dos *frames* exibidos no computador faz com que o componente de exibição de imagem emule a exibição de um vídeo. A Figura 7 ilustra esquematicamente como funciona a transmissão de imagens (captadas por uma *webcam*) entre o computador (servidor) e o outro computador (cliente) utilizando-se a internet.

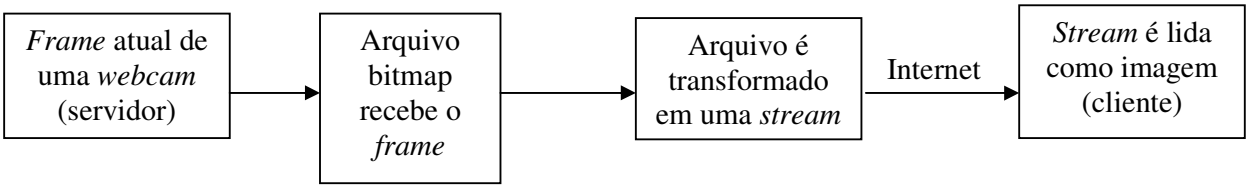


Figura 7 – Transmissão de informação de vídeo pela internet.

Uma vez estabelecida comunicação entre os computadores, de acordo com o processo na Figura 7, o usuário poderá visualizar as imagens na tela do computador (cliente) como se a *webcam* estivesse instalada nele próprio.

3. Descrição Geral do Projeto

3.1. O Sistema Produtivo considerado

O sistema produtivo considerado neste trabalho é um sistema de manufatura integrado por computador que executa o processo de montagem automática de diferentes peças (Festo, 1998). Cada peça é composta por: uma base, 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, como pode ser observado na Figura 8:

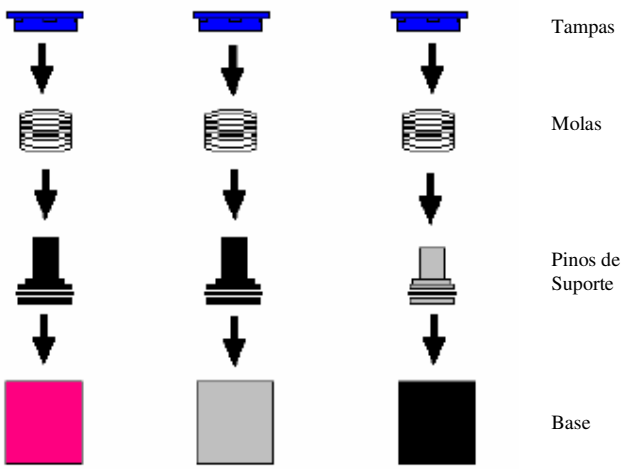


Figura 8 – Montagem dos diferentes cilindros pelo sistema CIM.

3.1.1. Descrição Geral

O sistema CIM é um equipamento para fins de treinamento especializado da empresa Festo que, na configuração disponível no Laboratório de Sistemas de Automação do PMR-EPUSP, é composto de 5 estações de trabalho, cada qual 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.

A Estação de Distribuição armazena as bases a serem montadas 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 vindas da estação de distribuição. Esta estação é também responsável por descartar peças rejeitadas neste teste. O Sistema Inteligente de Transporte conta com uma esteira de transporte e cinco carros (*pallets*).

Este sistema de transporte tem pré-definido quatro lugares distintos para a parada dos carros, sendo um deles destinado à Estação de Testes e outro à Estação de Montagem. O SIT destina-se a transportar as bases identificadas e testadas da Estação de Testes para a Estação de Montagem, onde a montagem do produto é finalizada. Os produtos finais são então transportados pelo SIT para uma outra localidade distinta.

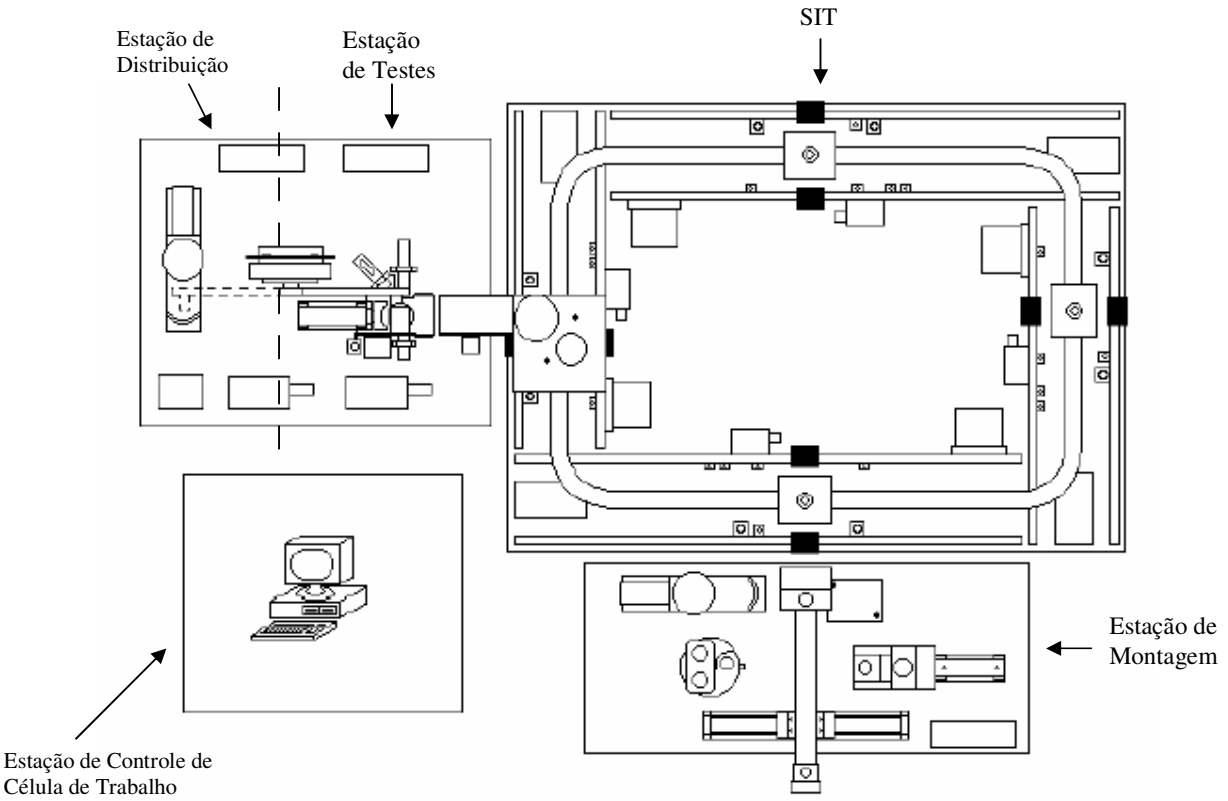


Figura 9 – Esquema ilustrativo do sistema CIM.

Entre as características do sistema CIM encontram-se:

- Estações de Distribuição e Testes: possui dispositivos para manipulação, testes de controle de qualidade através de sensores, aquisição e avaliação de sinais digitais e analógicos, assim como um sistema de controle para cada estação, com CLPs;

- Estação de Montagem: possui um manipulador de 3 eixos para a montagem das peças, sendo dois dos acionamentos por servo-motores (eixos x e y) e o terceiro por motor CC (eixo z), dotado ainda de dispositivo de garra dedicado. O controle é realizado por controlador programável.

- Sistema Inteligente de Transporte (SIT): visa o controle de fluxo de material e é composto de módulos de esteiras flexíveis, dotadas de dispositivos pneumáticos de fixação

de carros (*pallets*) e indexadores, que permitem o acesso para as estações de trabalho, além de futuras expansões.

- Estação de Controle de Célula de Trabalho: implementa o controle supervisório das demais estações e inclui um computador central (*Host Controller*) para controle do processo, com sistema de visualização do processo on-line.

A seguir apresenta-se uma descrição detalhada das Estações de Distribuição e Testes do sistema CIM, uma vez que estas são o material-base deste projeto.

3.1.2. Estação de Distribuição

A Estação de Distribuição pode ser definida como um sistema de alimentação cujas principais funções são:

- Retirar uma peça (base) do magazine de distribuição;
- Disponibilizar a peça para o processo subseqüente.

Na estação de distribuição, constam um compartimento de armazenagem de bases (magazine de distribuição), com sistema de retirada e posicionamento individual destas peças, e um mecanismo de transporte entre a estação de distribuição e a estação de testes (módulo de transferência).

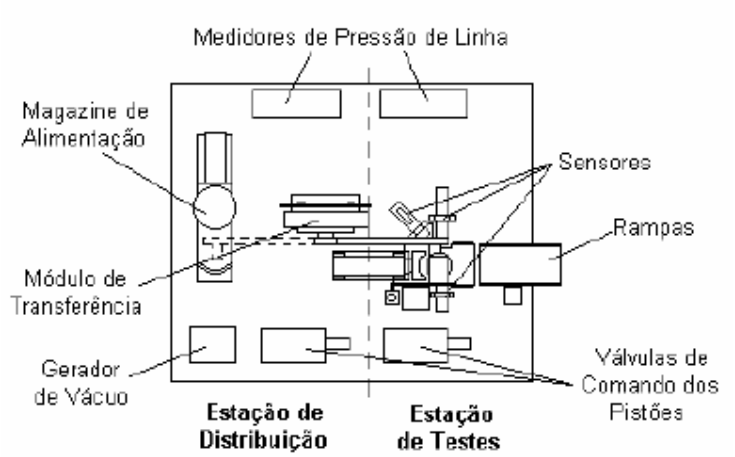


Figura 10 - Detalhamento dos componentes das Estações de Distribuição e Testes

O magazine de distribuição conta com um sensor de presença óptico no seu compartimento de armazenagem. Havendo peça neste compartimento, um feixe de luz é interrompido pela presença física da peça, impedindo que o mesmo alcance o sensor

(elemento foto-elétrico), que indica sua presença e sinaliza ao controle que o sistema pode iniciar a tarefa seguinte.

A retirada e o posicionamento individual das bases são realizados por um pistão pneumático, comandado por uma válvula de uma via. Este tipo de válvula apresenta somente um estado que pode ser mantido constantemente, isto é, ou haste do pistão está avançada (válvula fechada) ou recuada (válvula aberta). Neste dispositivo, o estado permanente da haste do pistão é a de avanço total. Quando é necessária a retirada de uma peça do compartimento de armazenagem e seu devido posicionamento, a válvula comuta seu estado e a haste do pistão recua, carregando, neste movimento, a primeira peça da fila (posição inferior da pilha de peças no compartimento de armazenagem) para a posição determinada, retornando em seguida a sua posição original. O controle dos limites de avanço e recuo da haste é realizado por sensores fim de curso magnéticos acoplados ao corpo do pistão.

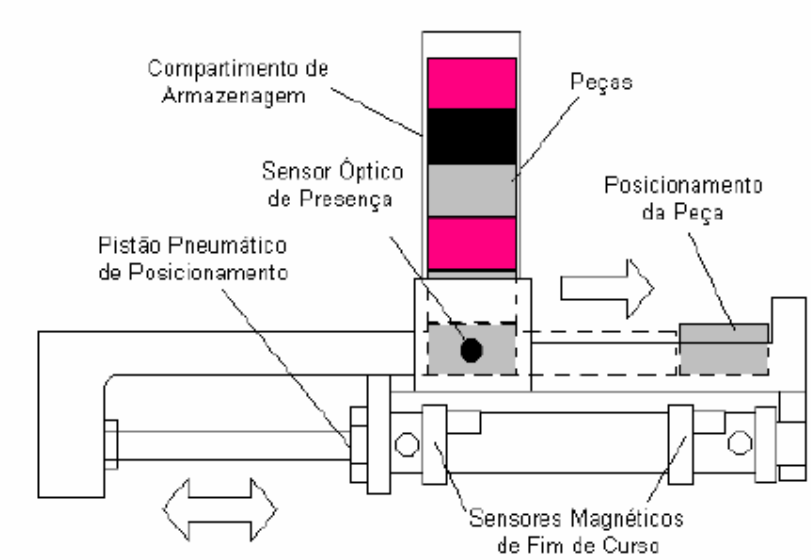


Figura 11 - Retirada e posicionamento de bases na estação de distribuição (magazine de distribuição)

O mecanismo de transporte entre as Estações de Distribuição e de Teste (módulo de transferência) é um braço articulado em uma de suas extremidades, cujo movimento se dá em plano vertical. O movimento deste braço mecânico é regulado por duas chaves fim de curso de roletes, vinculadas à rotação do mesmo em torno de sua extremidade articulada. Na sua extremidade livre, este mecanismo dispõe de uma "ventosa" capaz de fixar-se às peças através de vácuo, gerado externamente em uma unidade de geração de vácuo. Para a liberação da peça, basta eliminar o vácuo, permitindo-se contato das vias despressurizadas relacionadas a este mecanismo com o meio externo, eliminando a ação do vácuo gerado.

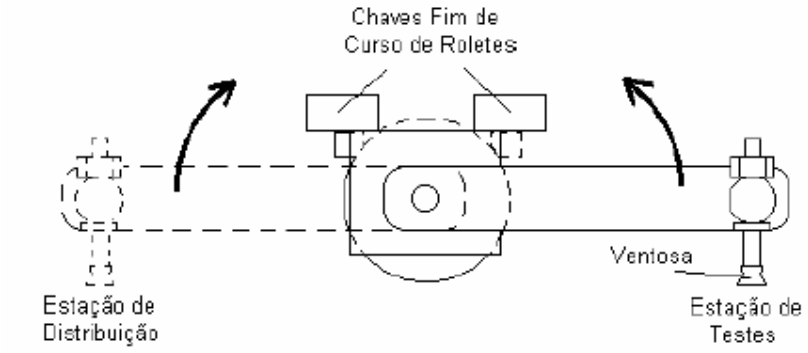


Figura 12 - Módulo de transferência

3.1.3. Estação de Testes

Tarefas importantes na execução de testes são a aquisição de informação e comparação de características específicas e, resultante disto, uma decisão entre "peça aceita" ou "peça rejeitada".

As funções da Estação de Testes são:

- Estabelecer as características do material da peça;
- Descartar ou disponibilizar a peça para a estação subseqüente.

Esta estação apresenta uma plataforma elevatória e diferentes tipos de sensores para a realização dos testes. Para tanto, esta estação define dois "andares". No "andar" inferior, existem três sensores distintos: indutivo, óptico e capacitivo, que estão dispostos de forma a não interferir na livre movimentação tanto da plataforma elevatória como do braço articulado do módulo de transferência (Estação de Distribuição). O sensor óptico é o único que permanece acoplado à plataforma elevatória, sendo os demais fixos na base da unidade. O sensor indutivo visa identificar peças metálicas (prateadas), o sensor óptico destina-se a identificar peças que refletem a luz por ele emitida (rosas ou prateadas), e o sensor capacitivo é utilizado para identificar a presença ou não de peças. Os três sinais emitidos pelos três sensores devem ser analisados em conjunto e permitem assim identificar a cor da peça presente na plataforma.

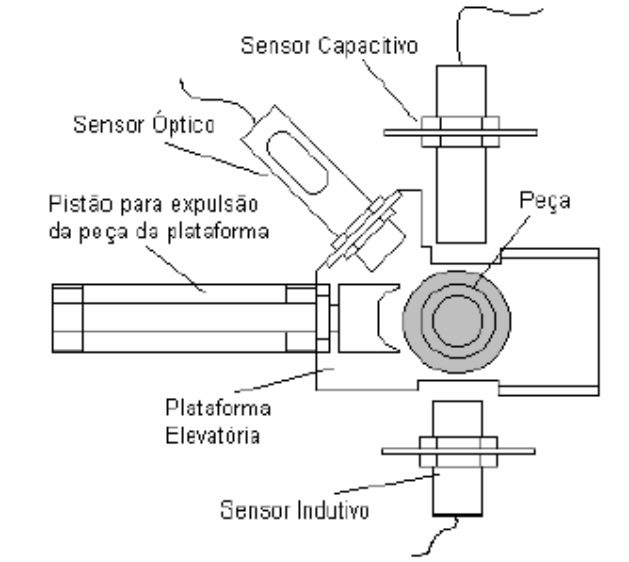


Figura 13 - Arranjo físico dos sensores para teste do tipo/material das peças na Estação de Testes

O "andar" superior, onde é realizado o teste de altura das peças, apresenta um pistão pneumático e um mecanismo acoplado à extremidade de sua haste, cujo movimento pressiona um sensor piezoelétrico acoplado à estrutura da plataforma elevatória, na posição vertical. O contato com a peça é realizado por uma pequena haste metálica, que é comprimida contra a peça a ser testada, em um movimento ao longo de seu eixo, gerando a compressão do elemento piezoelétrico. A compressão do elemento piezoelétrico gera uma corrente elétrica. A altura da peça pode ser, então, testada de acordo com a intensidade da corrente gerada deste contato (maior para compressões maiores -peças mais altas- ou menores para menores compressões - peças mais baixas), isto é, de acordo com a calibração deste sensor.

A plataforma elevatória conta com mais um pistão pneumático, na posição horizontal, que também se desloca junto com a plataforma elevatória, responsável por expulsar as peças, tanto aprovadas como reprovadas no teste de altura, através de rampas presentes em cada um dos andares: a rampa do "andar" superior para peças aprovadas e a do "andar" inferior para peças reprovadas. Para garantir a realização da tarefa, a ação deste pistão é temporizada, mantendo sua haste avançada por um tempo pré-determinado na programação, para somente então ocorrer o seu recuo, garantindo a expulsão da peça testada.

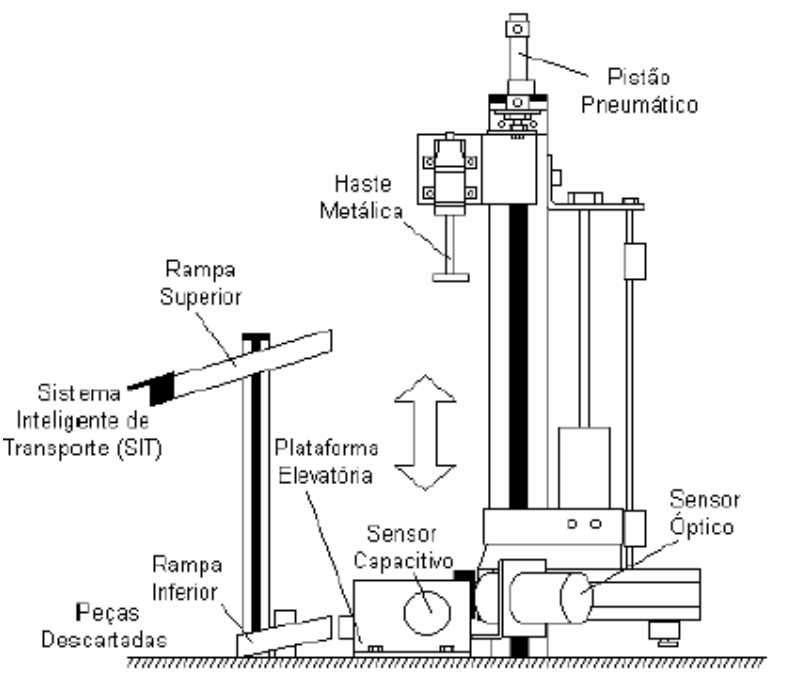


Figura 14 - Plataforma elevatória e rampas inferior e superior

Após a realização dos testes, as peças aprovadas seguem para um carro transportador (pallet) no sistema inteligente de transporte (SIT), por meio de uma rampa (superior), ou são descartadas no andar inferior, por meio de outra rampa (inferior).

3.1.4. Sistema de Controle

Cada estação de trabalho é controlada por meio de um controlador programável. O controlador que comanda a Estação de Distribuição é o FPC101-B, e o que controla a Estação de Testes é o FPC101-AF ambos da empresa Festo.

A ligação entre os elementos da bancada (dispositivos de atuação e dispositivos de detecção) e o controlador, em ambos os casos, é realizada através de dois terminais de entrada/saída, um deles ligado diretamente ao controlador e aos elementos da bancada, e outro ligado ao controlador e os dispositivos de comando e de monitoração (botões, chaves e lâmpadas sinalizadoras) existentes na estação. Cada qual possui 8 entradas por onde chegam as informações advindas dos dispositivos de controle e 8 saídas.

Os botões e chaves, que são únicos para as duas bancadas, possibilitam ativar o funcionamento das estações. Estes proporcionam a realização do processo de duas formas: a contínua, na qual o funcionamento das duas bancadas ocorre seqüenciada e ininterruptamente, e por etapas, onde se pode escolher qual das estações deseja-se ativar. Além disso, através dos botões e chaves pode-se interromper o processo a qualquer momento e retornar as estações às suas posições iniciais.

3.2. Considerações gerais sobre o projeto

O projeto aqui consiste em programar os CPs de maneira que as estações de trabalho executem o processo / tarefas previstas. Além disso, é necessário estabelecer a comunicação entre os CPs e os PCs (Estações de controle), isto é, programar a interface local e remota para que se comuniquem, respectivamente, com CP (serial) e PC Remoto (*sockets*) e PC Local (*sockets*).

A Figura 15 mostra uma representação esquemática das atividades desenvolvidas e os conceitos considerados.

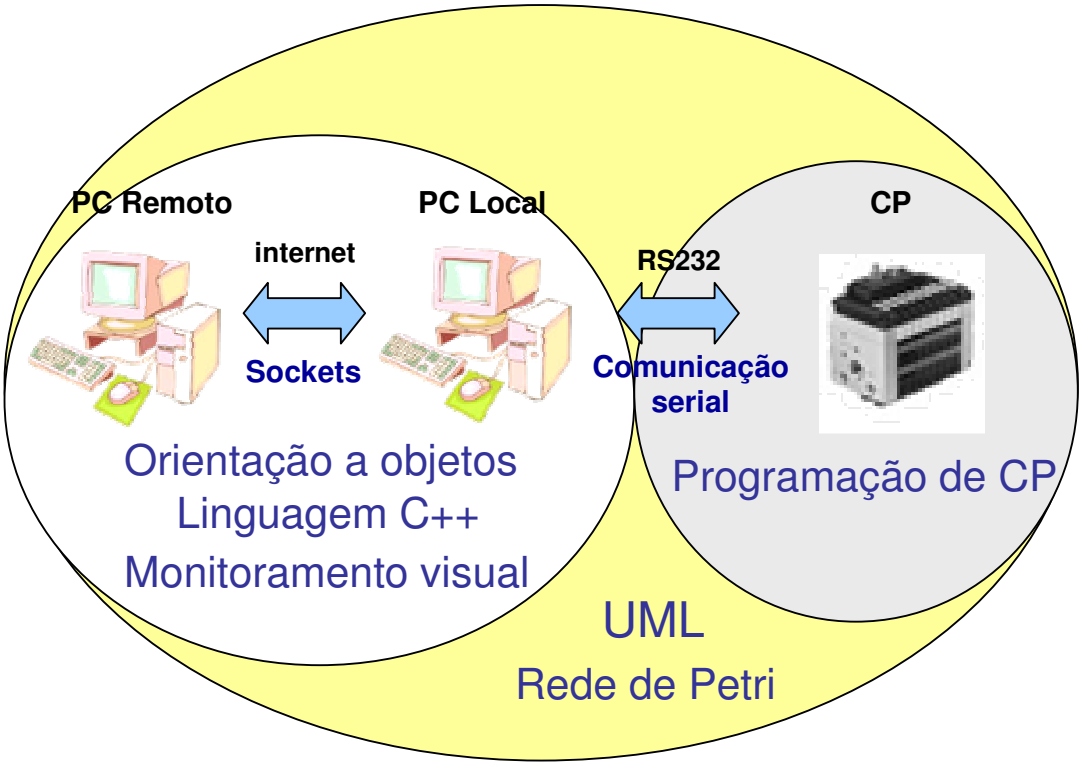


Figura 15 – Representação esquemática dos fundamentos aplicados ao projeto

4. Implementação

Para se implementar o comando e monitoramento das estações do sistema CIM foi adotada uma estratégia para elaboração do sistema de controle. A idéia é que uma vez visualizado o sistema, isto é, que se tenha definições e especificações precisas sobre sua estrutura, componentes físicos e lógicos e, processos envolvidos, bem como as maneiras como se deseja comandá-lo e monitorá-lo basta usar as ferramentas, conceitos e fundamentos adequados para implementar aquilo que foi planejado. Assim, torna-se chave a questão de como interpretar visualmente todas as possibilidades do seu sistema de controle. Neste projeto, isto foi realizado através dos diagramas UML e da representação do sistema CIM em redes de Petri, detalhados a seguir.

4.1. **Diagramas UML**

Inicialmente, estudou-se o sistema de controle através de digramas UML, a fim de se identificar todas as necessidades de interação entre sistema CIM, usuário local e usuário remoto. Os principais diagramas considerados são detalhados abaixo.

4.1.1. Diagrama de Casos de Uso:

Os atores do sistema em desenvolvimento são os usuários do programa. Inicialmente definiu-se três categorias de possíveis usuários, organizadas de forma hierárquica, de acordo com as funções disponibilizadas para o controle das estações de trabalho:

- EXPERT – Usuários desta categoria podem tanto realizar o processo continuamente (sem interrupções), quanto realizar o processo passo a passo (por etapas) ou realizar uma ou mais etapas do processo de forma isolada.

- INTERMEDIÁRIO - Usuários desta categoria podem realizar o processo continuamente ou passo a passo.

- LEIGO – Usuários desta categoria podem realizar o processo continuamente.

O diagrama de casos de uso resultante é apresentado na Figura 16:

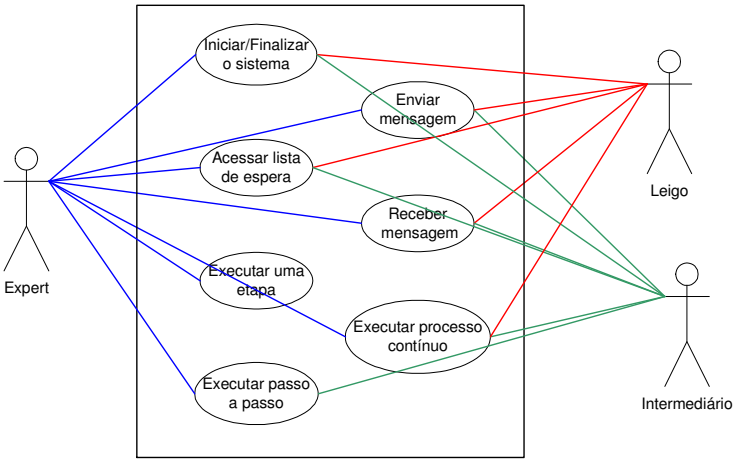


Figura 16 – Diagrama de casos de uso.

Nesta fase do projeto, para reduzir o tamanho do código de software, foi decidido que só haveria apenas um tipo de usuário, e este teria acesso a todo tipo de controle da estação (como seria o caso do usuário no topo da cadeia hierárquica pretendida).

4.1.2. Diagrama de Classes:

O ator externo "Usuário" solicita (via software) a execução de uma atividade para o *objeto* Proc_peça, que aciona os devidos *objetos* das estações de trabalho (os equipamentos, ou seja, os pistões, o braço mecânico e o gerador de vácuo) na ordem adequada para que a atividade possa ser concluída corretamente. Por exemplo, no caso das duas estações de trabalho consideradas tem-se o diagrama da Figura 17:

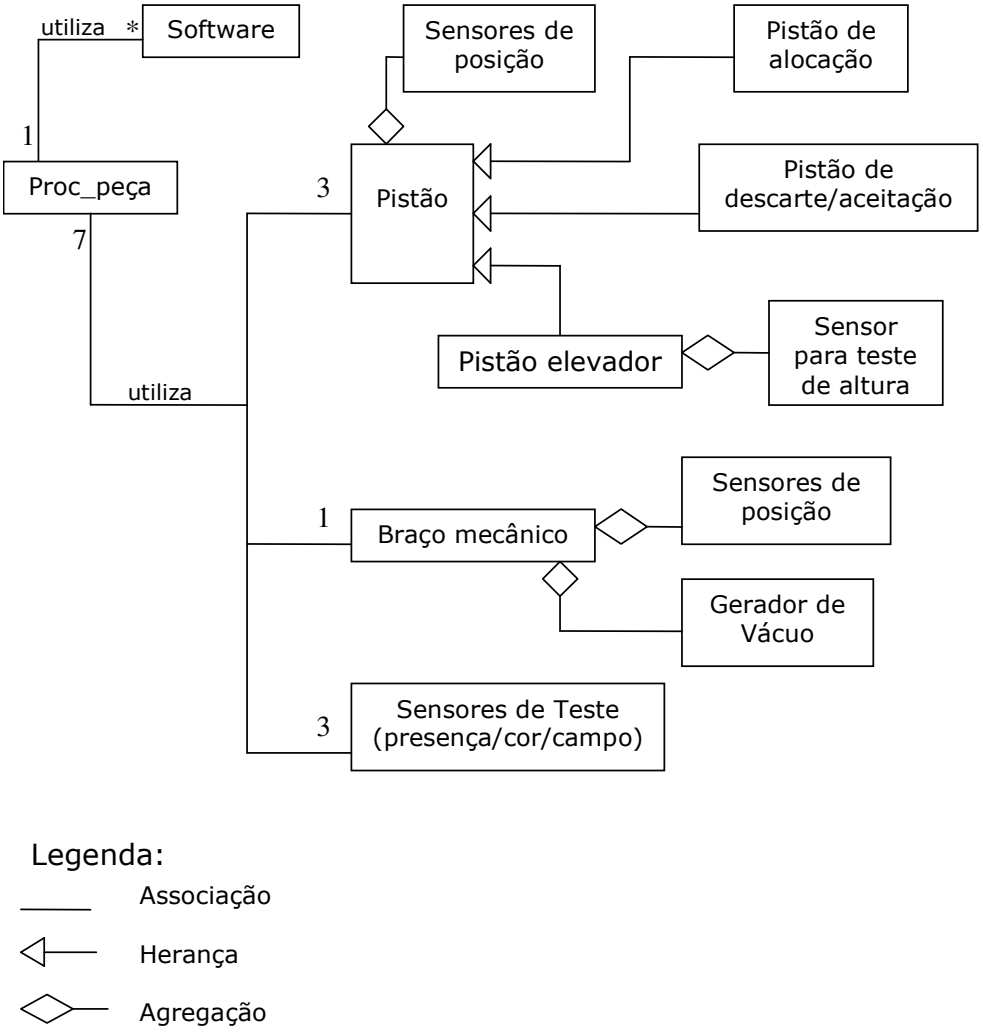


Figura 17 – Diagrama de classes.

4.1.3. Diagrama de atividades:

Os principais casos de uso são detalhados em diagramas de atividades.

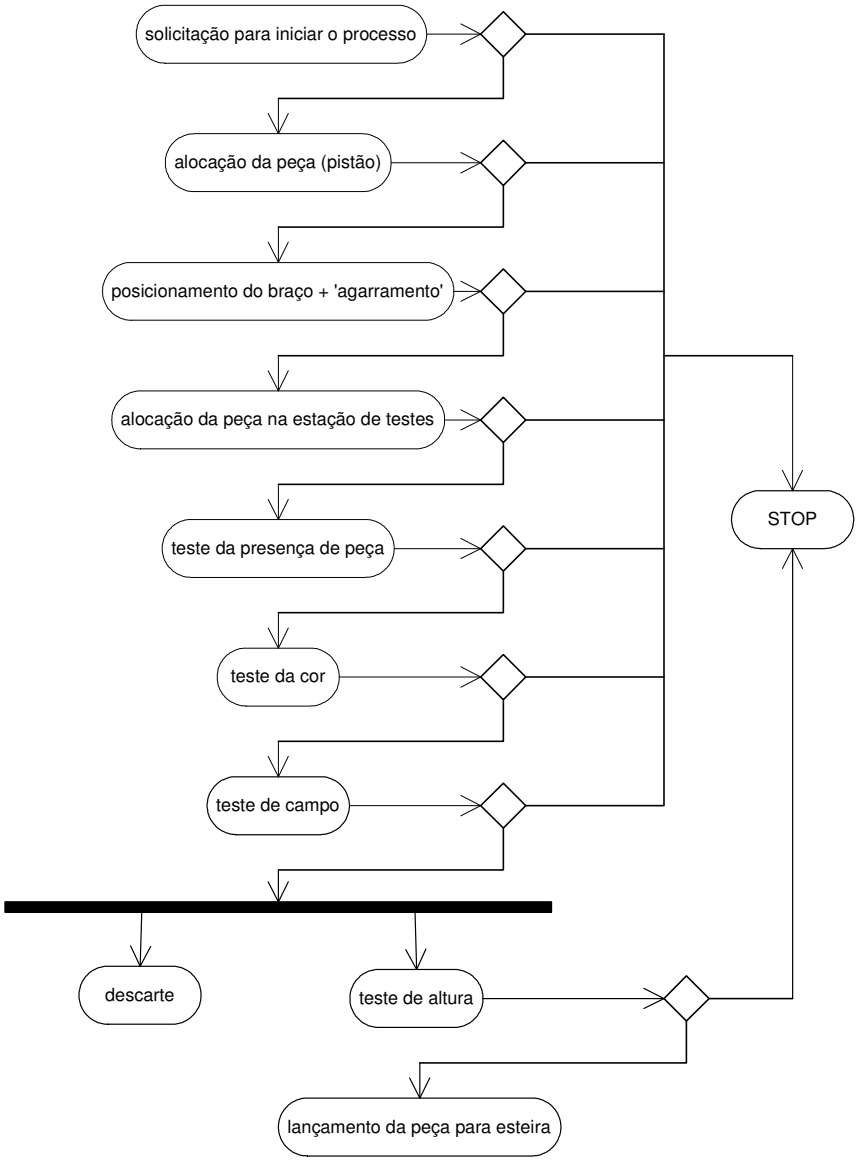


Figura 18 – Diagrama de atividades do caso de uso ‘Executar processo contínuo’.

No modo de execução de processo contínuo (Figura 18), as etapas do processo são realizadas em uma seqüência pré-estabelecida e, mesmo neste modo, é possível interromper o processo a qualquer momento clicando no botão STOP da interface principal (a partir daqui, esta será a referência para a tela principal do programa de controle desenvolvido). Durante a execução, o campo de status, localizado na interface principal, é continuamente atualizado para que o usuário possa monitorar o estado da estação de trabalho.

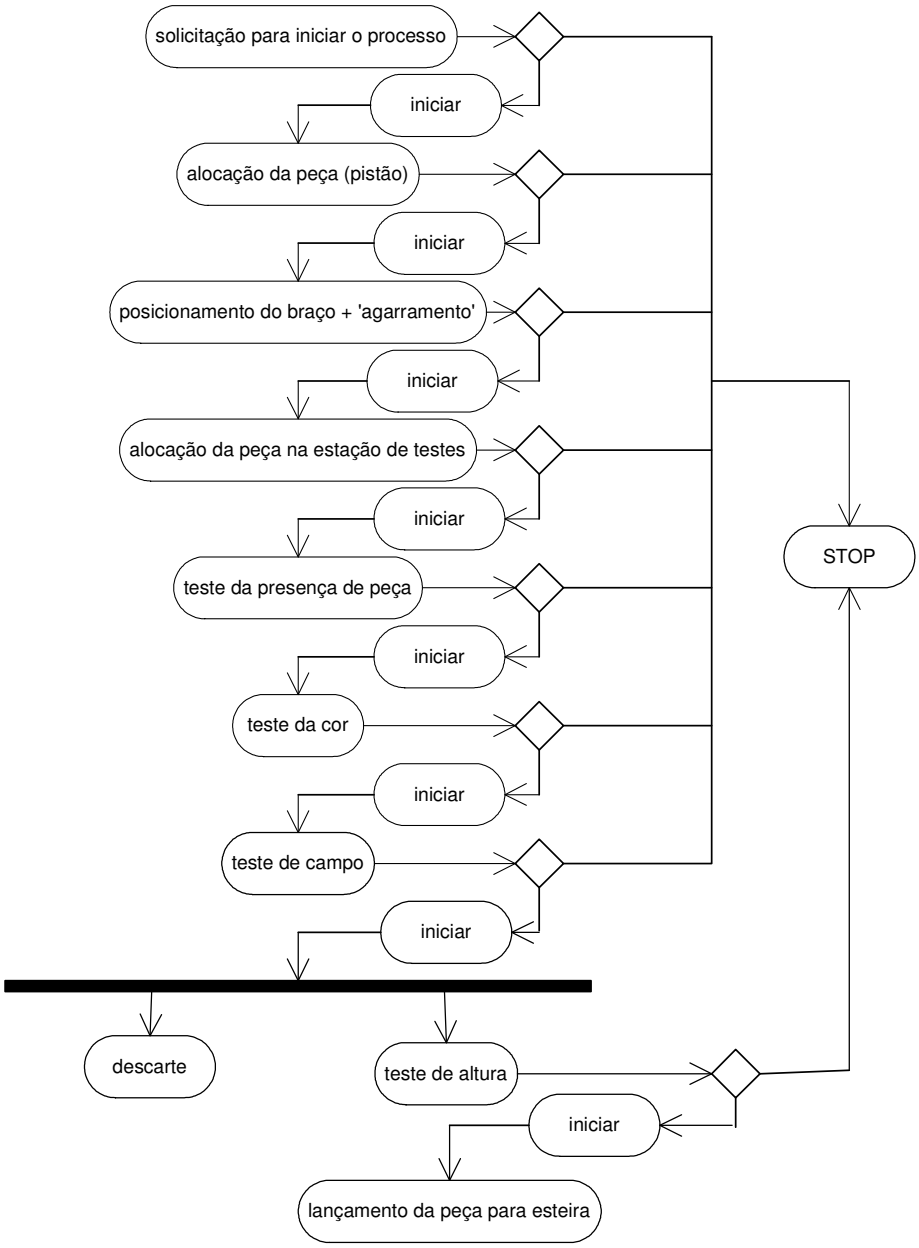


Figura 19 – Diagrama de atividades do caso de uso 'Executar processo por etapas'.

No modo de execução de processo por etapa (Figura 19), as etapas do processo também podem ser realizadas seqüencialmente, porém, é necessário que o usuário clique no botão 'Iniciar' para indicar a etapa a ser executada dentro da seqüência pré-estabelecida. O campo de status é atualizado após cada etapa. Clicando no botão STOP da interface principal pode-se interromper o processo a qualquer momento.

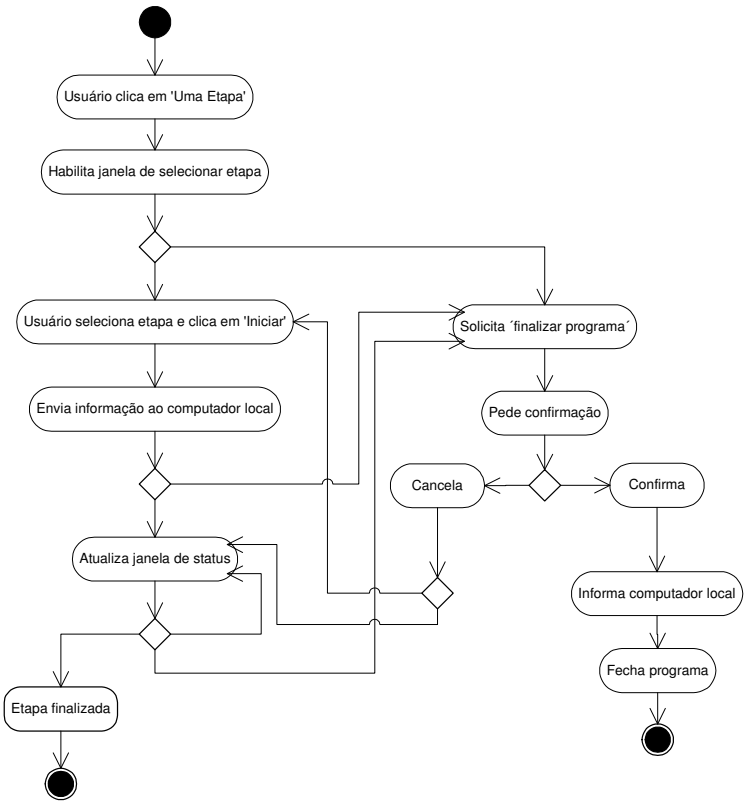


Figura 20 – Diagrama de atividades do caso de uso ‘Executar uma Etapa’.

No diagrama da Figura 20 a etapa selecionada pelo usuário é executada uma única vez. Durante a execução o campo de status é continuamente atualizado e o usuário pode solicitar a interrupção da etapa.

Após analisar todos os diagramas, ainda na fase inicial do projeto, optou-se por simplificar o sistema de controle. A principal simplificação foi eliminar a hierarquização de tipos de usuários. O estudo aprofundado de sockets foi fundamental para esta decisão uma vez que foi percebida a complexidade em se realizar uma conexão via internet em meio a códigos relativamente longos e complexos. Outras simplificações foram conseqüência da modelagem do sistema CIM através de rede de Petri.

4.2. Redes de Petri do sistema CIM

A modelagem de todo o sistema de controle foi feita, inicialmente, para a primeira estação de trabalho do sistema CIM, a estação de distribuição, e em seguida expandido para a estação de testes. O modelo de rede de Petri do funcionamento da primeira estação de trabalho considerada no projeto (Figura 21) foi decisivo na simplificação da interface principal do programa com o usuário. A partir da modelagem pode-se elaborar uma interface relativamente funcional, porém, sem alguns dos recursos periféricos planejados anteriormente nos diagramas UML.

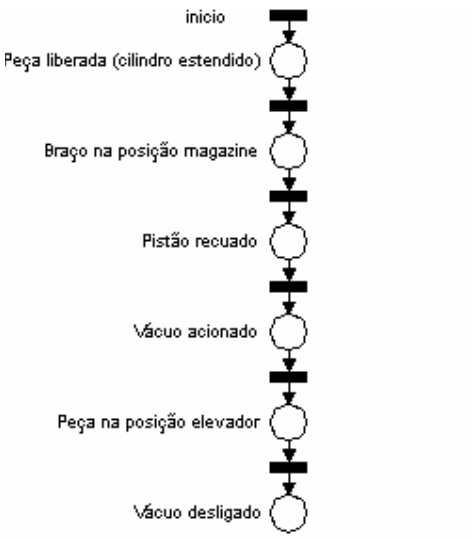


Figura 21 – Modelagem do funcionamento da estação de distribuição em rede de Petri

Para incluir o controle de outra estação de trabalho, isto é, além da estação de distribuição, o controle também da estação de testes, foi necessário expandir o programa responsável pelo comando e monitoração das estações. A expansão do programa segue a mesma lógica de programação anteriormente adotada estendendo as funções de comando e de monitoração. Para esta atividade, foi de fundamental importância o modelo em rede de Petri do sistema envolvendo as duas estações de trabalho (Figura 22).

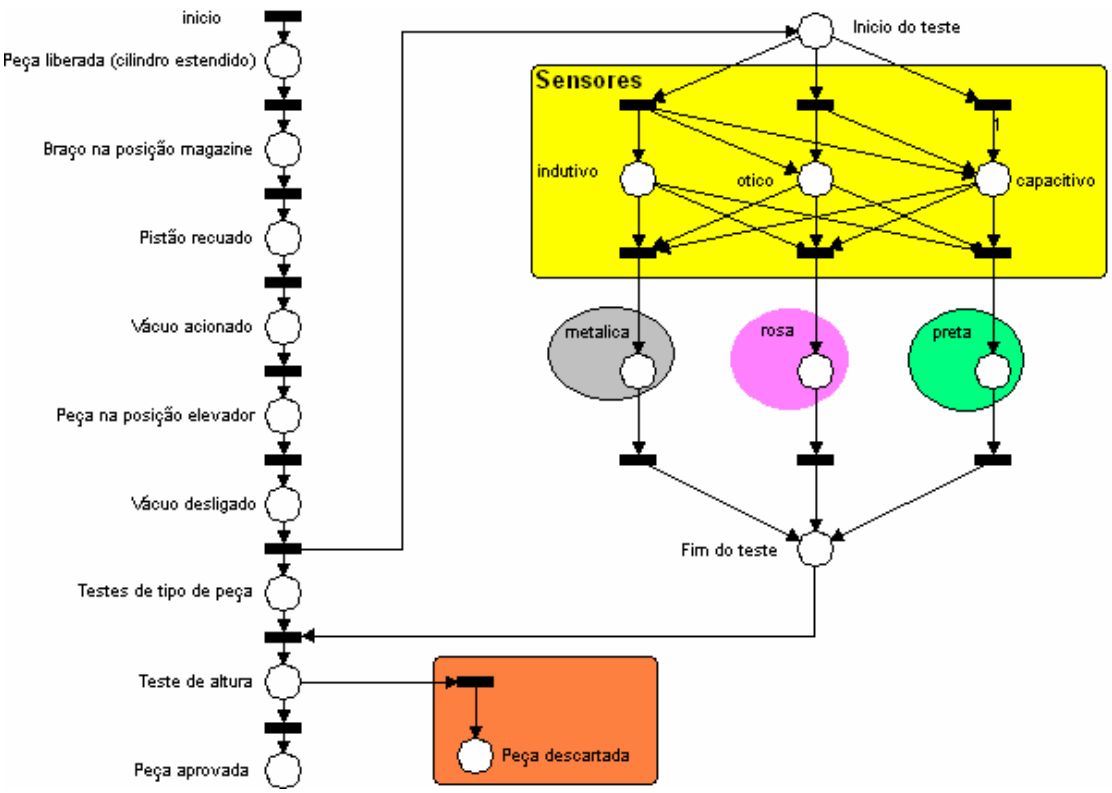


Figura 22 – Modelo do funcionamento conjunto das estações de distribuição e de testes em rede de Petri.

5. Testes e análise dos resultados

Os principais resultados reportados seguem a ordem de comunicação do sistema de controle. Inicialmente serão mostrados os resultados relativos à programação de CPs, seguidos pelos resultados de comunicação serial entre computador e CP, e, pela utilização de *sockets* para comunicar dois computadores através da *Internet*. Na seqüência, são mostrados os resultados da programação da interface final (tanto remota como local) para a primeira estação de trabalho (estação de distribuição), e, por fim, os resultados referentes à monitoração visual e transferência de vídeo pela *Internet*, para então, serem apresentados os resultados finais do projeto: a interface final do programa, já expandida para a estação de testes, e a sugestão de metodologia para desenvolvimento de sistemas de comando e monitoramento remoto de sistemas produtivos.

5.1. Programação do CP

A criação e implementação do programa do CP da estação de distribuição foi executada em diversas etapas. Após os estudos sobre lista de instruções e comunicação serial, foram analisadas possíveis formas de atualização do programa. A idéia inicial consistia em alterar apenas o trecho que carregava os valores de sinais de dispositivos de comando para uma variável indicativa de borda de subida, permanecendo o restante do código inalterado. A tentativa foi carregar neste trecho de código os sinais vindos da comunicação serial, substituindo os sinais originalmente vindos dos dispositivos de comando. Depois de definida esta estratégia, o próximo passo foi a familiarização com a implementação da comunicação serial através da lista de instruções. Ao fim desta etapa, foi possível concluir que a primeira estratégia traçada foi equivocada. Portanto, foi definida uma nova estratégia de alteração do programa, que embora mais trabalhosa devido ao fato de alterar significativamente o programa original, mostrou-se de implementação mais simples. Por fim, implementou-se todas as mudanças planejadas. A seguir, é detalhada cada uma das fases da concepção e implementação da atualização do programa no CP.

5.1.1. Primeira estratégia de atualização do programa do CP

No programa original do CP da estação de distribuição, existia uma rotina que cuidava da detecção de bordas de subida ocorridas devido ao acionamento dos botões da bancada. A

primeira idéia concebida foi a de alterar apenas esta rotina. Esta idéia fez-se válida, entre outros motivos, pelo fato de alterar apenas uma mínima parte do código original, reduzindo a propensão aos erros causados por alterações indevidas no programa original. A alteração idealizada é mostrada na Listagem 1 e Listagem 2:

```
STEP LOOP

IF                    NOP
 THEN  LOAD      (     IW1                    'Sinais dos botões
       EXOR            Old_IW1    )           'Sinais antigos
       AND             IW1
         TO            pEd_IW1                'Marcador de borda de subida

       LOAD            IW1
         TO            Old_IW1


 IF                    NOP
 THEN  JMP TO LOOP
```

Listagem 1 – Rotina original

```
STEP LOOP

IF                    NOP
 THEN  LOAD      (     Serial          'Porta serial
       EXOR            OldSerial )     'Porta serial (sinais antigos)
       AND             Serial
       TO            pEd_Ser           'Marcador de borda de subida

       LOAD            Serial
         TO            OldSerial

IF                    NOP
 THEN  JMP TO LOOP
```

Listagem 2 – Alteração na rotina original

Esta idéia foi abandonada após um maior entendimento do funcionamento da comunicação serial na linguagem do CP, pois o CP é capaz de responder a um simples *byte* enviado via porta serial, não havendo a necessidade de se trabalhar com operações que identifiquem as mudanças entre os sinais antigos e atuais. Fazendo uma simples correspondência entre os bytes recebidos e as ações executadas, é possível executar todas as rotinas de funcionamento.

5.1.2. Comunicação serial no CP

O software no qual o programa foi desenvolvido, o FST 4.11, disponibiliza diversos *drivers*, entre eles, o de comunicação serial. Ao adicionar este driver ao projeto, pode-se utilizar as rotinas pertencentes a este. Os comandos são tratados pelo software como CFMs,

ou seja, rotinas em outra linguagem às quais não se tem acesso ao código. Portanto, uma vez incluídos os comandos ao projeto, pode-se utilizá-los, porém não alterá-los.

A linguagem de lista de instruções oferece uma série de comandos relacionados à transmissão de dados através da comunicação serial. O programa desenvolvido para o CP da empresa Festo no controle das estações de distribuição e testes do sistema CIM conta com uma palavra ('*word*') que transmite os parâmetros necessários para este tipo de comunicação. Trata-se da FU32, variável interna utilizada para passar parâmetros para módulos. No caso de certos comandos, são necessárias outras palavras para que a informação seja transmitida, as quais não existem no programa atual da Festo, mas podem ser inseridas. Os comandos que possibilitam a transmissão de dados via comunicação serial, assim como os operandos necessários para tal, são listados a seguir:

- OPENCOM: abre a interface serial;

Parâmetro de Entrada: FU32 – interface serial

Parâmetro de Retorno: verdadeiro (0)/falso (1)

- CLOSECOM: fecha interface serial aberta;

Parâmetro de Entrada: FU32 – interface serial

Parâmetro de Retorno: verdadeiro (0)/falso (1)

- GETCOM: lê um caractere de uma interface serial;

Parâmetro de Entrada: FU32 – interface serial

Parâmetros de Retorno: verdadeiro (0)/falso (1)/nenhum dado recebido (-1)

FU33 – se FU32=0, FU33 recebe o caractere (0 a 255)

- PUTCOM: envia um caractere para uma interface serial;

Parâmetros de Entrada: FU32 – interface serial

FU33 – caractere a ser enviado (0 a 255)

Parâmetro de Retorno: verdadeiro (0)/falso (1)

A fim de compreender o funcionamento de cada um dos comandos, após integrá-los ao projeto, foi realizada uma série de testes, descritos a seguir:

- Teste 1: Enviar um *byte* via porta serial – Para realizar tal teste, foi utilizado o comando PUTCOM, definido no projeto como CFM4. Um programa simples em C++ (seção 4.2.1) foi desenvolvido para responder a este teste, mostrando um *label* caso o *byte* fosse recebido. Os códigos são mostrados na Listagem 3:

```
STEP INIT
 THEN  CMP 0                          'inicializa operandos

STEP
 THEN  LOAD           V1
         TO           FU32            'Parametros do PUTCOM
       LOAD           V6
         TO           FU33            'Parametros do PUTCOM

STEP
 THEN  CFM 4                          'PUTCOM
```

Listagem 3 – Teste 1

- Teste 2: Mover pistão através do comando do computador local – Neste teste, o *byte* correspondente ao movimento do pistão é ativado através de um programa em C++ que envia o comando de acordo com o acionamento do botão presente na interface do software do computador local. A recepção de sinal se dá através do comando GETCOM, definido como CFM3. Ao receber o *byte*, o programa do CP o carrega na palavra de saída correspondente aos atuadores da bancada, acionando o pistão. Os códigos programados são mostrados na Listagem 4:

```
STEP INIT
 THEN  CMP 0                          'inicialzia operandos

STEP
 THEN  LOAD           V1
         TO           FU32            'Parametro do GETCOM


STEP LOOP
 THEN
       CFM 3                          'GETCOM
 IF                   FU32
        =             V0
 THEN
```

```
       JMP TO 1
 OTHRW
       JMP TO LOOP

STEP 1
 THEN
       LOAD           FU33          'Parametro do GETCOM
         TO           OW0
```

Listagem 4 – Teste 2

- Teste 3: Como distinguir dois bytes recebidos – Adicionando-se outro botão à interface do software em C++, envia-se dois diferentes *bytes* ao CP. De acordo com o byte recebido, é carregado um valor diferente em uma palavra de saída. Os códigos programados são mostrados na Listagem 5:

```
STEP INIT
 THEN  CMP 0                         'inicializa operandos

STEP
 THEN  LOAD           V1
         TO           FU32

STEP LOOP
 THEN
       CFM 3                         'GETCOM
 IF                   FU32
         =            V0
 THEN
       JMP TO 0

 OTHRW
       JMP TO LOOP

STEP 0
 IF                   FU33
         =            V63
 THEN  JMP TO 1
 OTHRW
       JMP TO X

STEP X
 IF                   FU33
         =            V62
 THEN  JMP TO 2
 OTHRW
       JMP TO END

STEP 1
 THEN  LOAD           V6
         TO           OW1
       JMP TO END

STEP 2
 THEN  LOAD           V1
         TO           OW1

STEP END
 THEN  LOAD           V1
         TO           FU32
```

Listagem 5 – Teste 3

5.1.3. Reestruturação do programa do CP

A reestruturação do programa foi dividida em três casos: para o funcionamento automático, para o funcionamento por etapas e para o caso em que se deseja testar o funcionamento individual de qualquer dos componentes de uma estação de trabalho. Depois da implementação de cada caso, o próximo passo foi juntar todos os modos de funcionamento num único projeto, por meio da diferenciação dos *bytes* enviados conforme testes anteriores.

- Primeiro caso: Funcionamento automático – Adaptou-se do programa original do CP a rotina de *reset*, que posiciona a estação em seu estado inicial, e o programa que executa todos os procedimentos inerentes à estação de distribuição e de teste, em seqüência e automaticamente. Criou-se então o programa principal, que ao receber um *byte*, ativa o programa de *reset* e em seguida, o programa de controle que atua nas estações. Para que as duas rotinas não fossem executadas simultaneamente, um *flag* é levantado quando a rotina de *reset* é concluída, e a rotina seguinte só é executada caso este *flag* esteja levantado. Os códigos programados são mostrados na Listagem 6:

```
STEP INIT
 THEN  CMP 0                          'inicializa operandos


STEP MainLoop

 THEN
       LOAD           V1
         TO           FU32

STEP LOOP
 THEN
       CFM 3                          'GETCOM
 IF                   FU32
         =            V0
 THEN
       JMP TO 1
 OTHRW
       JMP TO LOOP

STEP 1
 THEN
       SET            P10            'P10: reset

STEP 2
 IF                   InitRdy        'F2.13: flag de Reset
 THEN
       SET            P11            'P11: sequência de eventos
```

Listagem 6 – Funcionamento Automático das estações de distribuição e de testes

- Segundo caso: Funcionamento por etapas – Neste projeto, foram criadas diversas rotinas, uma para cada etapa do processo, em substituição à rotina única do projeto

anterior. Para cada *byte* recebido no programa principal do CP, uma nova rotina é executada. Os códigos programados são mostrados na Listagem 7:

```
STEP INIT
 THEN  CMP 0                          'inicializa operandos


STEP MainLoop

 THEN
       LOAD           V1
         TO           FU32

STEP LOOP
 THEN
       CFM 3                          'GETCOM
 IF                   FU32
         =            V0
 THEN
       JMP TO 1
 OTHRW
       JMP TO LOOP

STEP 1
 THEN
       SET            P10             'P10: reset

STEP 2
 THEN
       CFM 3                          'GETCOM
 IF                   FU32
         =            V0
 THEN
       JMP TO 3
 OTHRW
       JMP TO 2

STEP 3
 THEN
       SET            P12             'P12: Passo 1

STEP 4
 THEN
       CFM 3                          'GETCOM
 IF                   FU32
         =            V0
 THEN
       JMP TO 5
 OTHRW
       JMP TO 4

STEP 5
 THEN
       SET            P13             'P13: Passo 2

STEP 6
 THEN
       CFM 3                          'GETCOM
 IF                   FU32
         =            V0
 THEN
       JMP TO 7
 OTHRW
       JMP TO 6

STEP 7
 THEN
       SET            P14             'P14: Passo 3

STEP 8
 THEN
       CFM 3                          'GETCOM
```

```
 IF                  FU32
        =            V0
 THEN
       JMP TO 9
 OTHRW
       JMP TO 8

STEP 9
 THEN
       SET           P15           'P15: Passo 4

STEP 10
 THEN
       CFM 3                       'GETCOM
 IF                  FU32
        =            V0
 THEN
       JMP TO 11
 OTHRW
       JMP TO 10

STEP 11
 THEN
       SET           P16           'P16: Passo 5

STEP 12
 THEN
       CFM 3                       'GETCOM
 IF                  FU32
        =            V0
 THEN
       JMP TO 13
 OTHRW
       JMP TO 12

STEP 13
 THEN
       SET           P17           'P17: Passo 6

STEP 14
 THEN
       CFM 3                       'GETCOM
 IF                  FU32
        =            V0
 THEN
       JMP TO 15
 OTHRW
       JMP TO 14

STEP 15
 THEN
       SET           P18           'P18: Passo 7

STEP 14
 THEN
       CFM 3                       'GETCOM
 IF                  FU32
        =            V0
 THEN
       JMP TO 15
 OTHRW
       JMP TO 14

STEP 16
 THEN
       SET           P19           'P19: Passo 8

STEP 15
 THEN
       CFM 3                       'GETCOM
 IF                  FU32
        =            V0
 THEN
       JMP TO 16
 OTHRW
       JMP TO 15
```

```
STEP 16
 THEN
       SET             P20             'P20: Passo 9
STEP 17
 THEN
       CFM 3                           'GETCOM
 IF                    FU32
         =             V0
 THEN
       JMP TO 18
 OTHRW
       JMP TO 17

STEP 18
 THEN
       SET             P21             'P21: Passo 10

STEP 19
 THEN
       CFM 3                           'GETCOM
 IF                    FU32
         =             V0
 THEN
       JMP TO 20
 OTHRW
       JMP TO 19

STEP 20
 THEN
       SET             P22             'P22: Passo 11

STEP 21
 THEN
       CFM 3                           'GETCOM
 IF                    FU32
         =             V0
 THEN
       JMP TO 22
 OTHRW
       JMP TO 21

STEP 22
 THEN
       SET             P23             'P23: Passo 12
```

Listagem 7 – Funcionamento por etapas das estações de distribuição e de testes

- Terceiro caso: Teste dos atuadores – De forma semelhante ao segundo caso, foram criadas rotinas específicas para cada tipo de teste que se deseje realizar. A principal diferença deste caso para o caso anterior é que neste existe a necessidade de diferenciar os *bytes* recebidos do computador, para que se possa definir qual teste deve ser executado. Os códigos programados para a estação de distribuição são mostrados na Listagem 8:

```
STEP INIT
 THEN  CMP 0                           'inicializa operandos

STEP MainLoop

 THEN
       LOAD            V1
         TO            FU32

STEP LOOP
 THEN
       CFM 3                           'GETCOM
 IF                    FU32
         =             V0
```

```
 THEN
       JMP TO 1
 OTHRW
       JMP TO LOOP

STEP 1
 THEN
       SET             P10             'P10: reset

STEP 2
 THEN
       CFM 3                           'GETCOM
 IF                    FU32
         =             V0
 THEN
       JMP TO 3
 OTHRW
       JMP TO 2

STEP 3
 IF                    FU33
         =             V10
 THEN
       SET             P12             'P12: Teste 1
       JMP TO 2
 OTHRW JMP TO 4

STEP 4
 IF                    FU33
         =             V20
 THEN
       SET             P14             'P14: Teste 3
       JMP TO 2
 OTHRW JMP TO 5


STEP 5
 IF                    FU33
         =             V30
 THEN
       SET             P13             'P13: Teste 2
       JMP TO 2
 OTHRW JMP TO 6


STEP 6
 IF                    FU33
         =             V40
 THEN
       SET             P16             'P16: Teste 5
       JMP TO 2
 OTHRW JMP TO 7


STEP 7
 IF                    FU33
         =             V50
 THEN
       SET             P15             'P15: Teste 4
       JMP TO 2
 OTHRW JMP TO 8


STEP 8
 IF                    FU33
         =             V63
 THEN
       SET             P17             'P17: Teste 6
       JMP TO 2
```

Listagem 8 – Funcionamento do teste dos atuadores da estação de distribuição

5.2. Comunicação Serial em C++

A partir do código fornecido em (Traverse, 2007), foram adaptados primeiramente os programas de teste que interagem com os programas-teste do CP. Com o funcionamento destes, foi possível adaptar a interface elaborada no projeto, habilitando a comunicação serial ao software final do projeto.

5.2.1. Testes em C++

O código obtido exemplifica a comunicação via serial a partir de dois *memos*. O primeiro programa-teste adaptado deste código possui um botão que envia um *byte* e um *label* invisível que aparece no programa do CP quando este recebe um *byte*. Para os demais testes, foram adicionados mais botões, de acordo com as necessidades dos programas-teste do CP (no caso do teste para diferenciar *bytes* recebidos, eram necessários dois botões, cada qual enviando um *byte* diferente para o CP; no caso do teste dos atuadores eram necessários sete botões, cada qual enviando um diferente *byte* para o CP). As diferenças concentram-se essencialmente em dois eventos: o "clique" do botão e o evento DisplayIt(), que trata do recebimento dos dados. As modificações são apresentadas na Listagem 9:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  byte = '?';
 // TRANSMITE O BYTE ESCOLHIDO.
  TransmitCommChar(hComm, byte);
}

void __fastcall TRead::DisplayIt(void)
{
Form1->Label1->Show();
}
```

Listagem 9 – Alterações dos Códigos para Testes em C++

Com a verificação do funcionamento dos programas de teste em comunicação em C++, aplicaram-se os mesmos conceitos na interface do software do projeto.

5.3. Programação de conexão à Internet via sockets

Os dois testes mais relevantes para o projeto na fase de estudos de *sockets* foram: a implementação dos programas (cliente e servidor) que trocassem informação (1 byte) entre si, pela rede ao clicar-se em um botão; e a implementação dos programas (cliente e servidor) que realizassem um simples *chat* entre si.

TESTE 1:

Troca de informação:

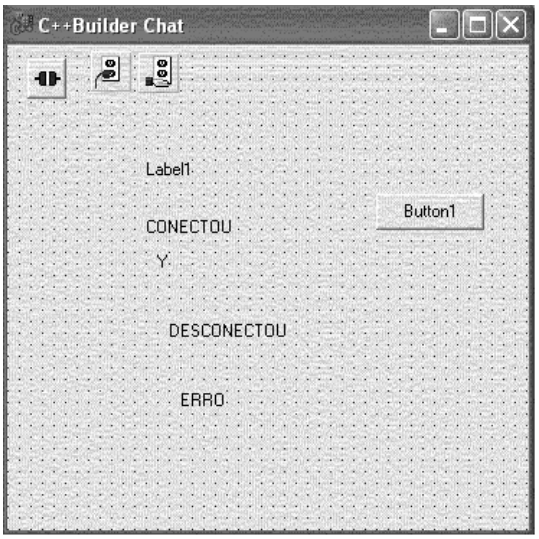


Figura 23 –Tela do programa cliente

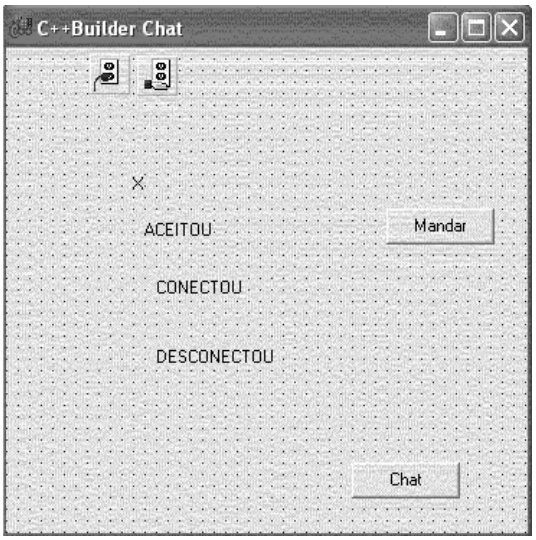


Figura 24 – Tela do programa servidor

O Cliente (Figura 23): Os *objetos* importantes do cliente e suas atividades vinculas aos *métodos* usados são:

- Botão de Conexão (canto superior esquerdo da tela):

Método OnClick :

```
ClientSocket->Host    =    "192.168.0.244";    //    define    o    IP    do    servidor
ClientSocket->Active = true;       // ativa o objeto ClientSocket
```

- ClienteSocket (ao lado do Botão de Conexão):

Método ClientSocketConnect:

```
Form1->Label2->Show();   // mostra Label CONECTOU
```

Método ClientSocketDisconnect:

```
Form1->Label4->Show();    // mostra Label DESCONECTOU
```

Método ClientSocketError:

```
Form1->Label5->Show();       //mostra       Label       ERRO,       relativo       à
//impossibilidade de conexão
```

Método ClientSocketRead:

```
if ( Socket->ReceiveText() == "Y"){   // se recebeu via sockets Y

              Form1->Label3->Show();    // mostra Label Y na tela

      }
```

- Botão de Enviar (Button1):

Método OnClick :

```
ClientSocket->Socket->SendText("X");   // manda X via sockets
```

O Servidor (Figura 24): Os *objetos* importantes do servidor e suas atividades vinculas aos *métodos* usados são:

- Botão de Enviar (botão 'Mandar' da tela):

Método OnClick :

```
ServerSocket1->Socket->Connections[0]->SendText("Y");        //        manda        Y
// via sockets
```

- ServerSocket (ao lado esquerdo do botão 'Mandar'):

Método ServerSocketAccept:

```
Form1->Label2->Show();  // mostra Label ACEITOU
```

Método ServerSocketClientConnect:

```
Form1->Label3->Show();    // mostra Label CONECTOU
```

Método ServerSocketClientDisconnect:

```
Form1->Label4->Show();    // mostra Label DESCONECTOU
```

Método ServerSocketClientRead:

```
if (Socket->ReceiveText() == "X")  {   // se recebeu via sockets X

     Form1->Label1->Show();       // mostra na tela Label Y

 }
```

Feito este teste, pode-se entender os princípios da comunicação via *sockets* e desenvolver um programa que realizasse as funções básicas de um *chat* entre 2 usuários e a troca de códigos (X e Y por exemplo).

TESTE 2:

Chat:

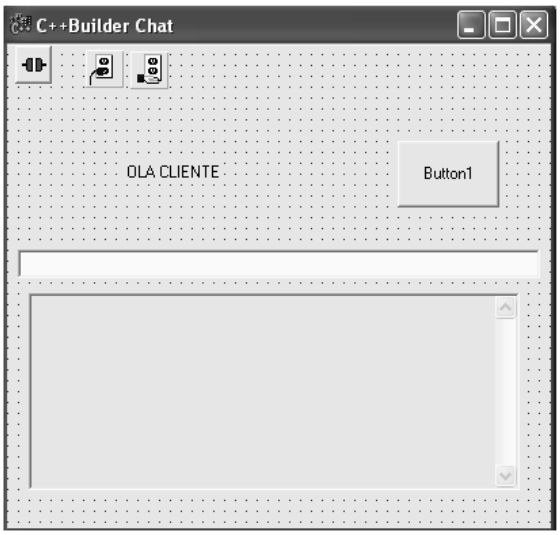


Figura 25 – Tela do programa cliente

Figura 26 –Tela do programa servidor

O Cliente (Figura 25): Os *objetos* importantes do cliente e suas atividades vinculadas aos *métodos* usados são:

- Botão de Conexão (canto superior esquerdo da tela):

Método OnClick :

```
ClientSocket->Host = "192.168.0.244";   // define o IP do servidor     ClientSocket-
>Active = true;       // ativa o objeto ClientSocket
```

- ClienteSocket (ao lado do Botão de Conexão):

Método ClientSocketConnect:

```
Memo2->Lines->Add("PC LOCAL CONECTADO..."); // coloca no quadro de mensagens   a
//frase entre aspas duplo
```

Método ClientSocketDisconnect:

```
Form1Memo2->Lines->Add("PC LOCAL DESCONECTADO!!!");

Memo2->Lines->Add("\r\n"); //depois de colocada a frase acima, pula linha
```

Método ClientSocketError:

```
Memo2->Lines->Add("Error connecting to: PC LOCAL"); // adiciona…
```

Método ClientSocketRead:

```
c=Socket->ReceiveText(); //uma AnsiString recebe o que chega pela rede

                if (c == "Y") {            // se for Y

                        Label1->Show(); //mostre a mensagem de boas //vindas
```

```
                }
                else {          // caso contrario, o que quer que seja
                        Memo2->Lines->Add("PC LOCAL: " + c);// ponha //no chat
                 }
```

- Botão de Enviar (Button1 da tela):

Método OnClick :

```
ClientSocket->Socket->SendText("X");   // manda X via sockets
```

- Edit (barra para escrever o texto a ser mandado):

Método :

```
if (Key == VK_RETURN)    //se apertou enter
        {
                Memo2->Lines->Add("PC REMOTO: " + Edit1->Text);  //escreva no seu quadro
de mensagens a mensagem digitada
                ClientSocket->Socket->SendText(Edit1->Text);// //mande para o servidor
                Edit1->Clear();  //limpe  o  campo  onde  você  acabou  //de  escrever  uma
mensagem
        }
```

O Servidor (Figura 26): O raciocínio é análogo, porém no servidor não existe botão de conexão e ao criar o Form já ativa-se o servidor com o comando:

```
ServerSocket->Active = true;
```

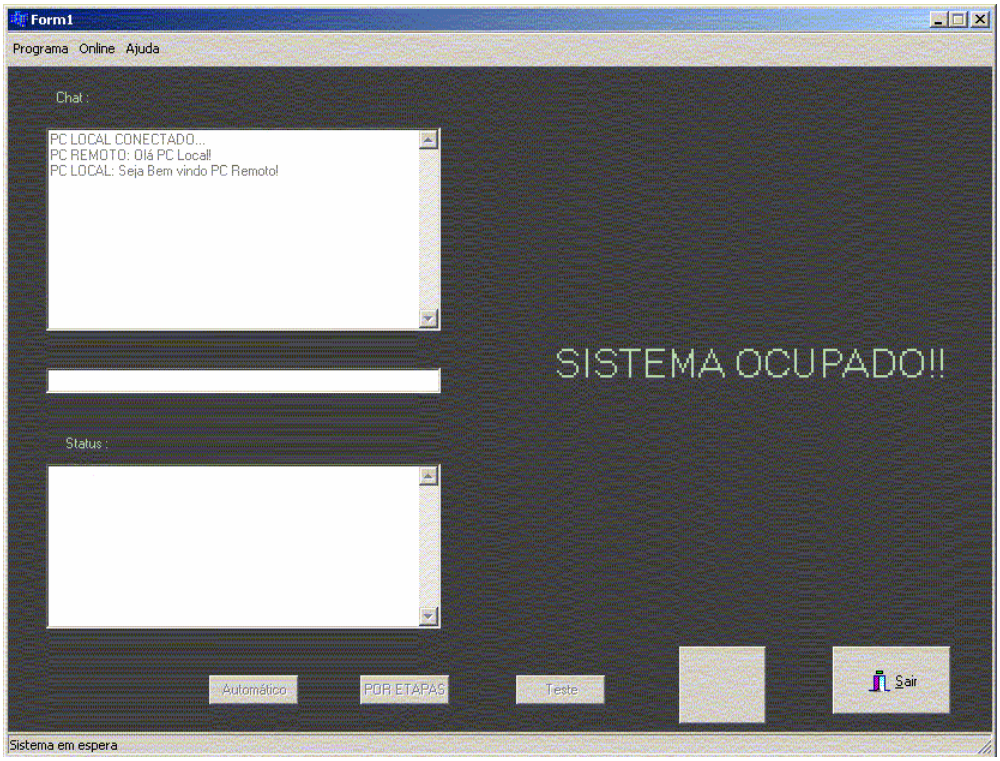
5.4. Programação da interface de controle da estação de distribuição

Entendido como realizar comunicação via *sockets*, foi possível programar a interface parcial do projeto, somente referente ao controle da estação de distribuição.

Comunicação pela Internet: O princípio básico de conexão foi que para estabelecer conexão, o servidor deve permitir o controle remoto (quando o usuário remoto se conectar com o servidor, caso esteja liberada a conexão remota o cliente receberá um byte para entender que pode controlar a estação) caso contrário o controle estará restrito ao PC local. Os comandos passados de cliente para servidor (e vice-versa) são códigos em forma de bytes interpretados e traduzidos nos programas. Caso a informação recebida não corresponda a nenhum código (e, portanto a nenhum comando ou resposta) esta deverá ser impressa no *chat*. As possíveis situações de conexão já na interface final são:

Situação 1:

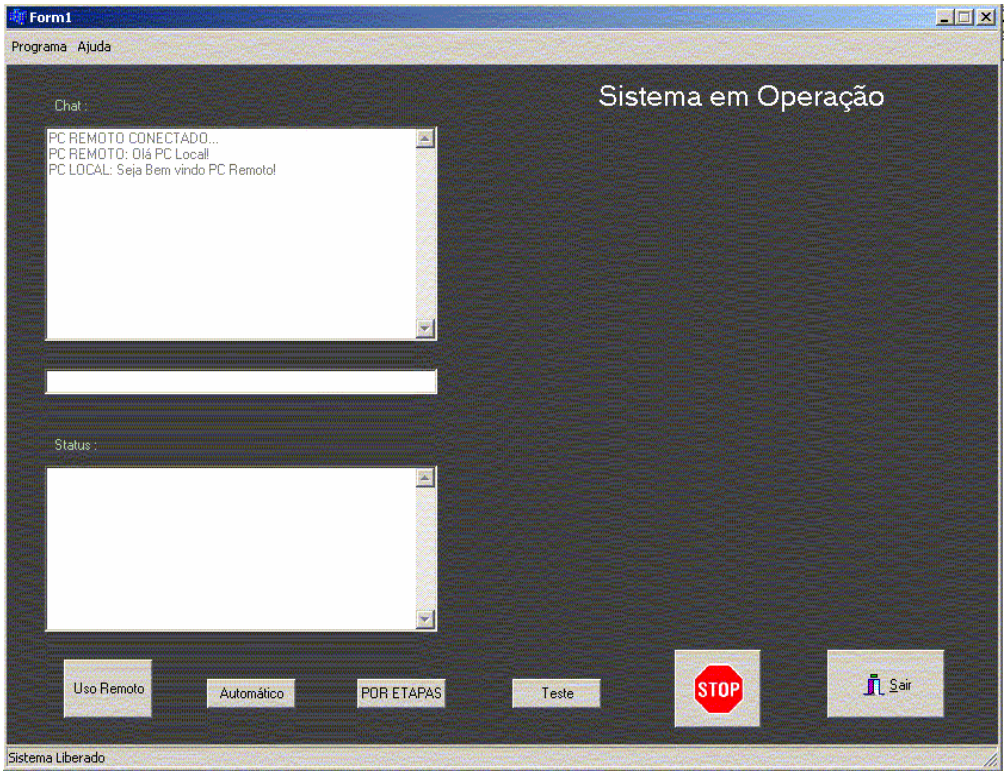
O cliente não consegue acessar o controle da estação de trabalho porque este está sendo usado pelo PC local (designação do computador que se encontra fisicamente no mesmo local das estações de trabalho, o PC remoto é aquele que se conecta ao PC local via *Internet* e pode se encontrar em qualquer localidade), mas mesmo assim, o cliente está conectado ao servidor e pode conversar pelo *chat* com o mesmo (Figura 27):



TELA DO PC REMOTO

Figura 27 – Cliente não consegue acessar a bancada.

Neste caso, o PC local está com pleno acesso e pode realizar os comandos desejados, além da possibilidade de conversar com o usuário remoto (Figura 28):



TELA DO PC LOCAL

Figura 28 – Servidor conectado e operando.

O Status Bar e o quadro de mensagens (Figura 29) fornecem informações relevantes tais como “conexão estabelecida”, “aguardando conexão”, “Usuário Remoto desconectado!” para acompanhar a evolução da conexão e do processo (alem de dois Labels auxiliares).

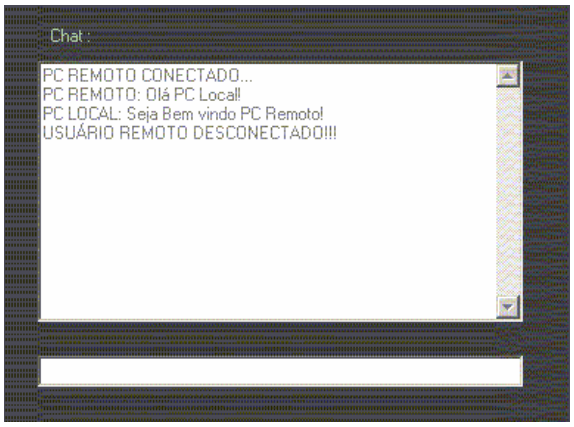


Figura 29 – *Chat* no detalhe

Situação 2:

O PC local pode permitir o uso remoto através do botão “Uso Remoto”. Se há cliente *online*, este automaticamente passa a ter o controle da estação, se não há ainda, assim que um cliente se conectar ao servidor ele automaticamente estará com o controle da estação de trabalho (a não ser que o PC local pare de permitir uso remoto neste ínterim, o que recairia na situação anterior) (Figura 30):

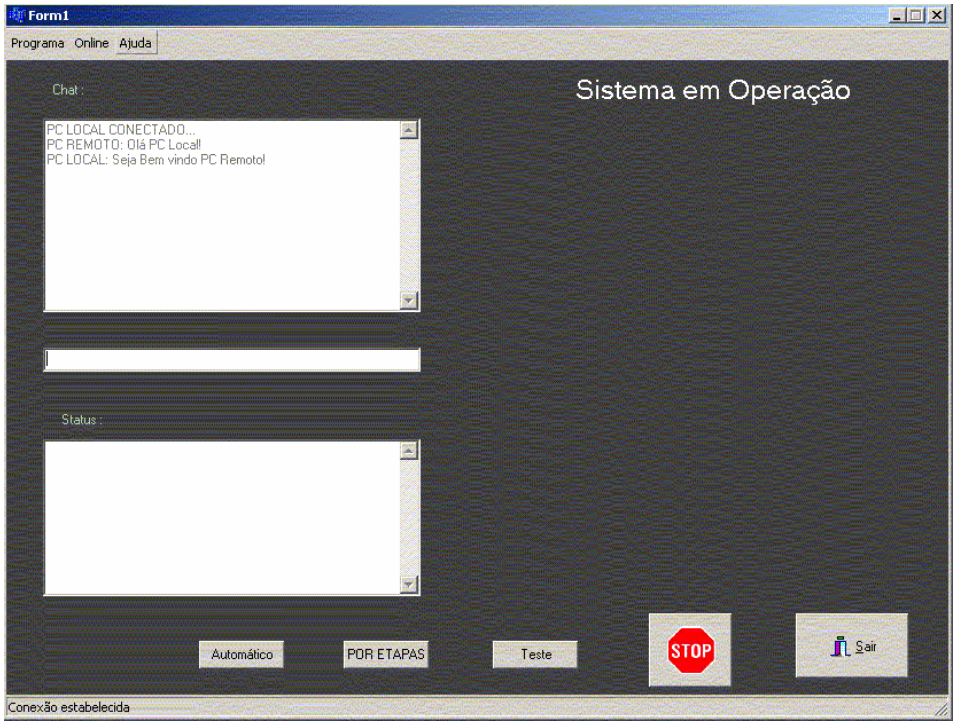
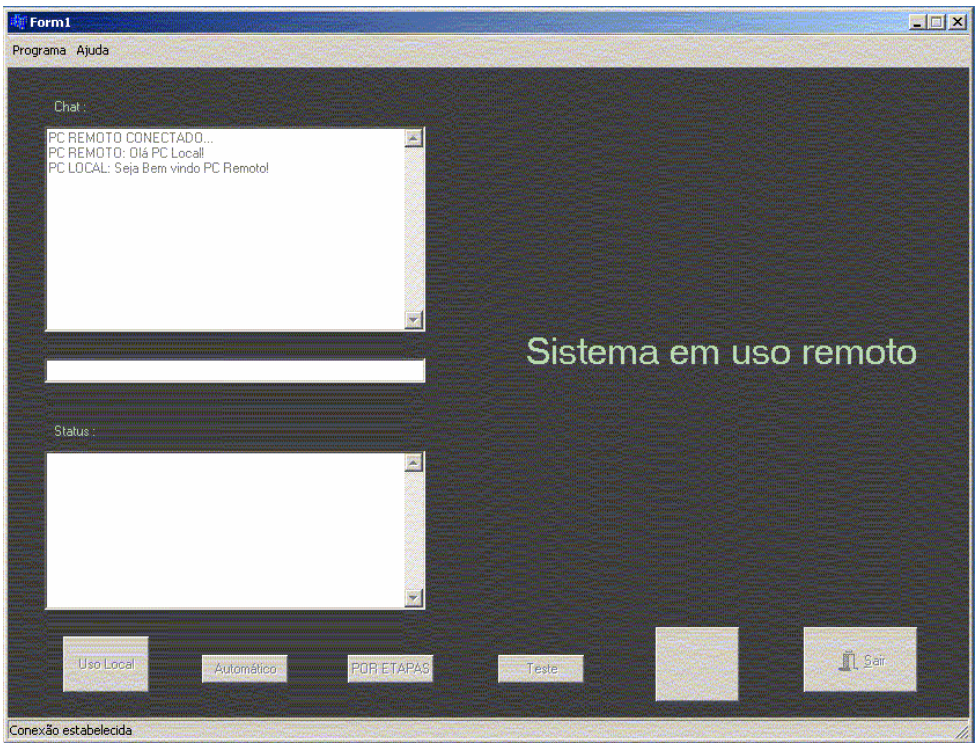


Figura 30 –Clienteconectado e operando.

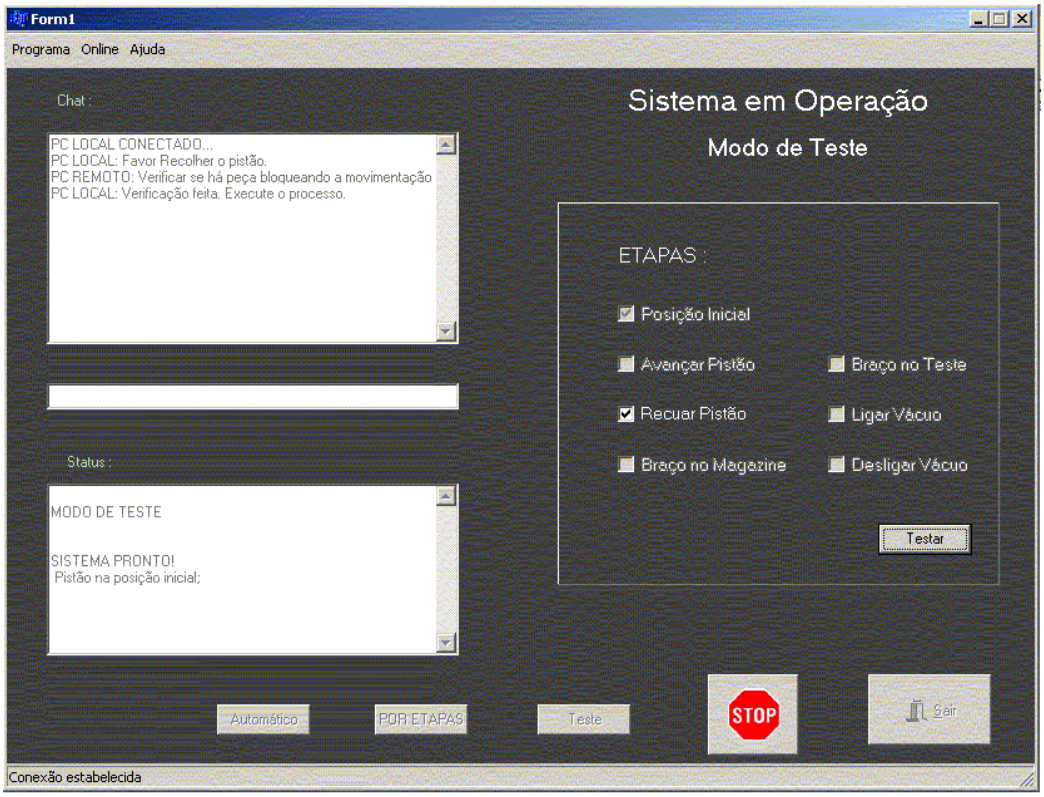
Depois da conexão do usuário remoto, o usuário local deverá esperar o cliente se desconectar para voltar a ter controle sobre a estação de trabalho. Lembrando que o *chat* está sempre funcionando enquanto cliente e servidor estão *online* (Figura 31):



TELA DO PC LOCAL

Figura 31 – Servidor *online*, mas sem o controle da estação.

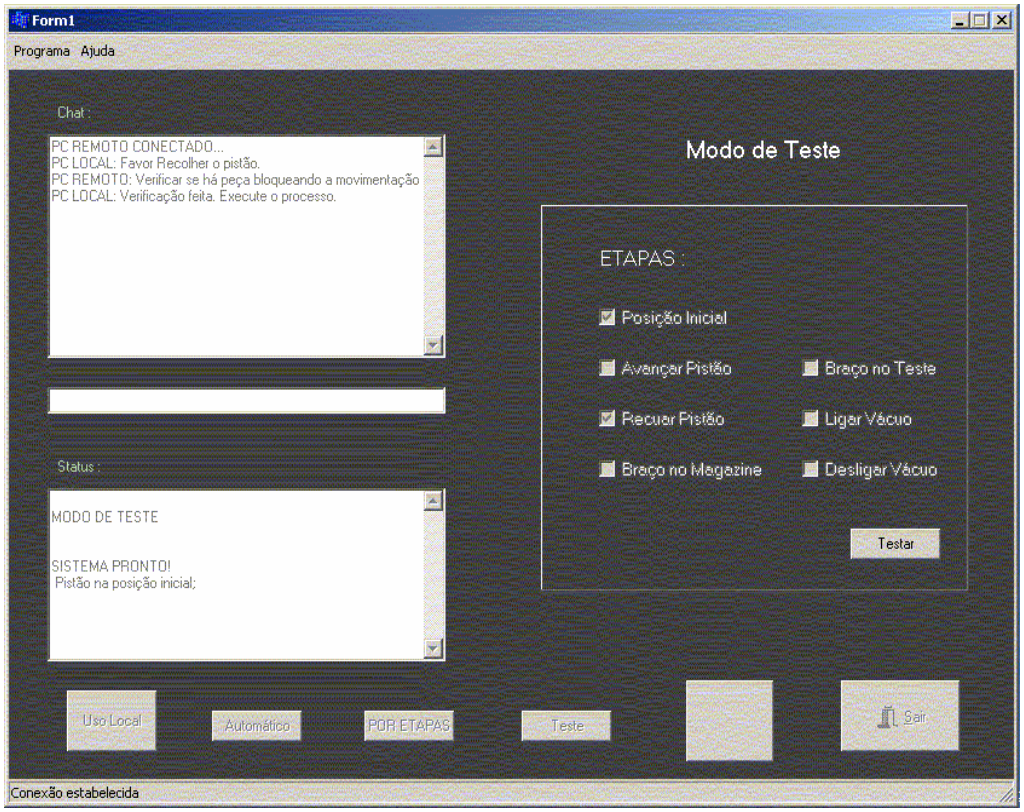
Com base no mecanismo de conexão entre servidor e cliente, anteriormente citado, apresenta-se agora a interface final funcionando para alguns casos específicos (Figura 32):



TELA DO PC REMOTO

Figura 32 – Cliente escolhe uma etapa e realiza teste

Enquanto o cliente realiza o processo de teste (durante todo processo que qualquer um dos usuários estiver realizando, este só pode acionar o botão de STOP), o servidor apenas acompanha a evolução dos comandos do cliente (e enquanto o usuário remoto não se desconectar, o servidor estará com sua tela inteira, a exceção do *chat*, desabilitada) (Figura 33):



TELA DO PC LOCAL

Figura 33 – Servidor acompanha Modo Teste (Remoto executando)

Implementado o controle e monitoração da primeira bancada, para atingir os objetivos do projeto ainda resta a expansão do controle para a estação de distribuição e o aprimoramento das interfaces finais (dos PCs local e remoto) através da adição de um campo onde serão mostradas as imagens do sistema CIM adquiridas por uma *webcam*.

5.5. Monitoração via *webcam*

Foi desenvolvido um programa de teste para a monitoração de estações de trabalho através de uma *webcam* instalada no PC Local. O programa envolve três elementos para a interação com o usuário: 2 botões e uma "janela" de vídeo. Ao se acionar um dos botões (*Button1*) a "janela" de vídeo é ativada e o as imagens da *webcam* aparecem nesta "janela", o segundo botão (*Button2*) ao ser acionado desativa a "janela" de vídeo e a transmissão das imagens é interrompida. As Figuras 34, 35 e 36 ilustram o funcionamento deste programa.

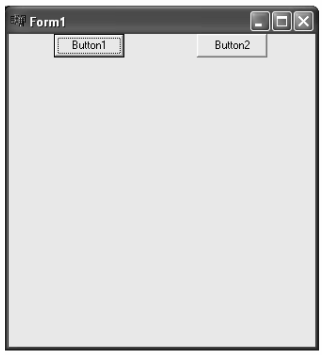


Figura 34 – Início.

Figura 35 – Após acionar o *Button1*.

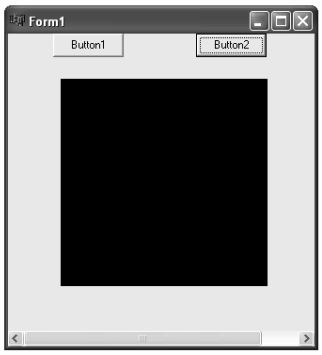


Figura 36 – Após acionar o *Button2*.

O comando ao se acionar o *Button1* que é executado é:

```
Video1->Enabled = true;            // Video1 é a janela de vídeo
```

Já o comando ao se acionar o *Button2* que é executado é:

```
Video1->Enabled = false;
```

5.6. Transferência de imagens pela Internet

Diversos programas de teste foram concebidos e implementados para estudar a transferência de imagens pela Internet, dentre estes programas são apresentados a seguir aqueles mais significativos no sentido que forneceram subsídios para a implementação realizada.

Teste 1 : Salvando *frames* em um arquivo BITMAP

O programa cuja tela é exibida na Figura 37 tem como propósito salvar cada *frame* capturado pela *webcam* em um arquivo BITMAP.

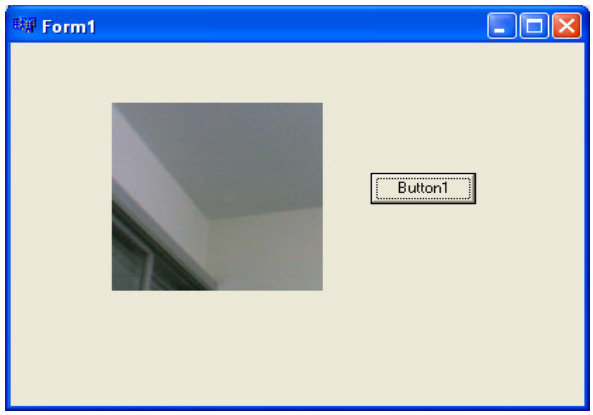


Figura 37 –Tela do primeiro programa de teste

O componente VIDEO CAPTURE é ativado quando se aciona o botão *Button1*. No evento *OnFrame*, que é acionado a cada novo *frame* capturado pela *webcam*, executa-se a seguinte linha de comando:

```
Video1->SaveFrameToFile('f');
```

Assim, cada novo *frame* será salvo no arquivo f.

Teste 2 : Usando o componente de imagem

Este programa é similar ao anterior, porém, além do componente VIDEO CAPTURE é utilizado o componente IMAGE. Assim como no caso anterior, os *frames* capturados pela *webcam* são salvos em um arquivo BITMAP, porém, além disso, esse mesmo arquivo é carregado no componente de imagens toda vez que o *frame* vindo da *webcam* é alterado. O comando responsável por essa ação também é executado no evento *OnFrame* e corresponde à seguinte linha de comando:

```
Image1->Picture->Bitmap->LoadFromFile('f');
```

A tela desse programa pode ser vista na Figura 38:

Figura 38 –Tela do segundo programa de teste

Esse programa demonstra que no cliente as imagens capturadas na *webcam* do servidor podem ser carregadas a partir de um arquivo BITMAP ou uma *stream* correspondente ao mesmo. Para tanto basta que este arquivo ou esta *stream* chegasse no próprio cliente via *sockets*.

Teste 3 : Usando a Internet

Conhecendo-se como processar as informações de vídeo tanto no servidor como no cliente, deve-se implementar a comunicação entre ambos para viabilizar a troca de imagens pela Internet. Assim, a Figura 39 mostra a tela do programa servidor (aquele que deve ser executado no computador com a *webcam*) e a Figura 40 a tela do programa cliente (que deve receber e carregar as imagens).

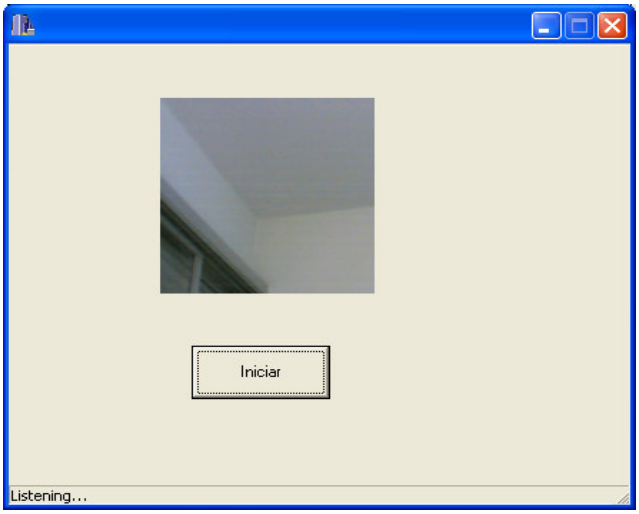


Figura 39 – O servidor mostrando as imagens da *webcam*

A diferença deste programa (servidor) para os anteriores é o fato deste possuir um objeto (não-visível) responsável por enviar *streams* via *sockets.* Trata-se do objeto NMStrm que tem como propriedades básicas o IP do computador que deverá receber as *stremas* enviadas e a porta pela qual este programa deverá enviar as mesmas.

Assim, no evento *OnFrame*, este programa realiza duas outras tarefas além de salvar cada novo *frame* no arquivo BITMAP. São elas: criar uma *stream* que recebe o arquivo de imagem e enviar esta *stream* via *sockets*. As linhas de código para realizar tais novas tarefas são:

```
frame = new TFileStream('f', fmShareDenyNone);
NMStrm1->PostIt(frame);
```

O cliente por sua vez deve contar com o objeto (não-visível) NMStrmServ responsável por manipular as *streams* recebidas via *sockets*. No evento *OnMsg* deste novo objeto as

streams recebidas são processadas. As linhas de código referentes ao recebimento da *stream* e de sua exibição no componente de imagem são:

```
frame = strm;   // ao disparar, o evento OnMsg guarda a stream recebida na variável strm
Image1->Picture->Bitmap->LoadFromStream(frame);
```

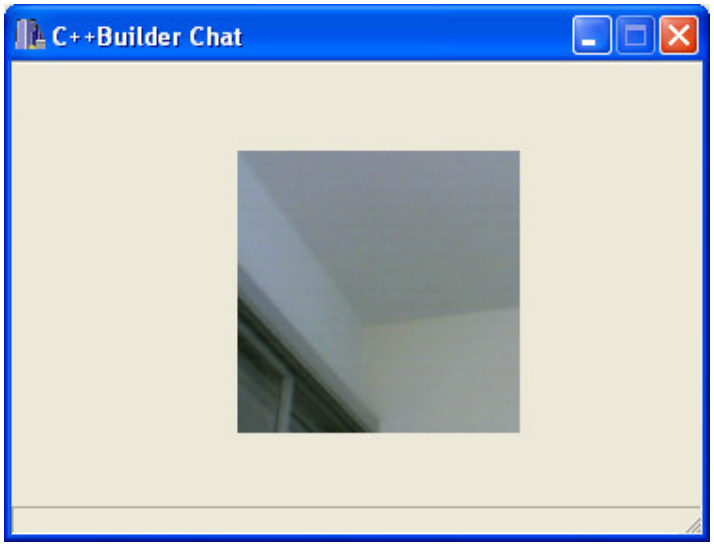


Figura 40 – O cliente reproduzindo as imagens da *webcam*

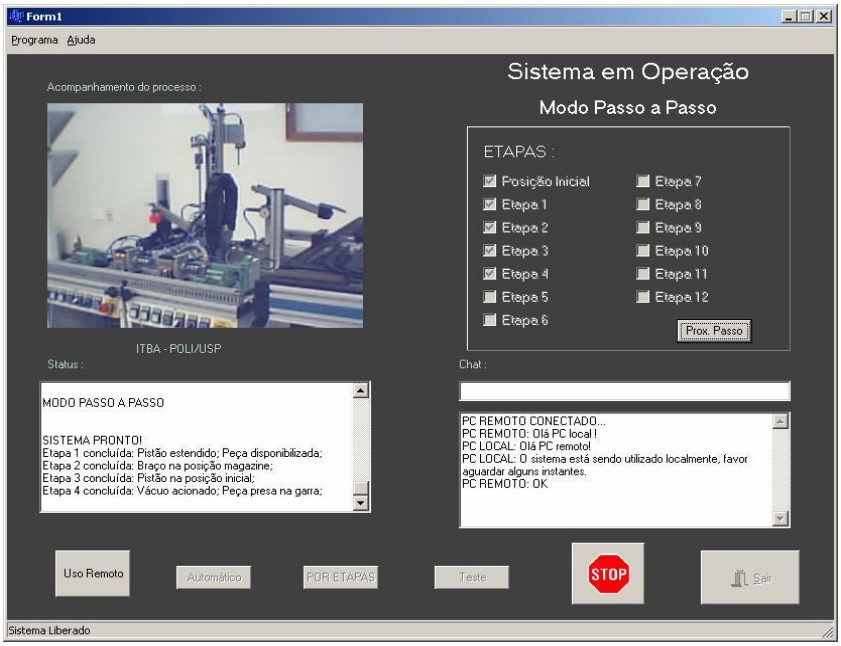
Desta forma alcançou-se à meta de transmitir informações de vídeo pela Internet, utilizando-se do princípio de que diversos *frames* reproduzidos em seqüência emulam um vídeo. Vale lembrar que quanto mais eficiente for a conexão com a Internet melhor será o resultado do vídeo reproduzido no cliente.

5.7. Interfaces finais no computador do usuário do SP

A interface desenvolvida anteriormente (tanto no PC local como no PC remoto) da estação de distribuição, foi aprimorada e expandida para incluir o comando e monitoração da estação de teste. O aprimoramento do programa segue a mesma lógica de programação anteriormente adotada estendendo, evidentemente, as funções de controle e de monitoração. A tela de interface do PC local e a do PC foram redesenhadas para incluir um componente de vídeo. Seguem abaixo as Figuras 41 a 46 com interfaces no PC local e no PC remoto durante a execução de diversas tarefas:

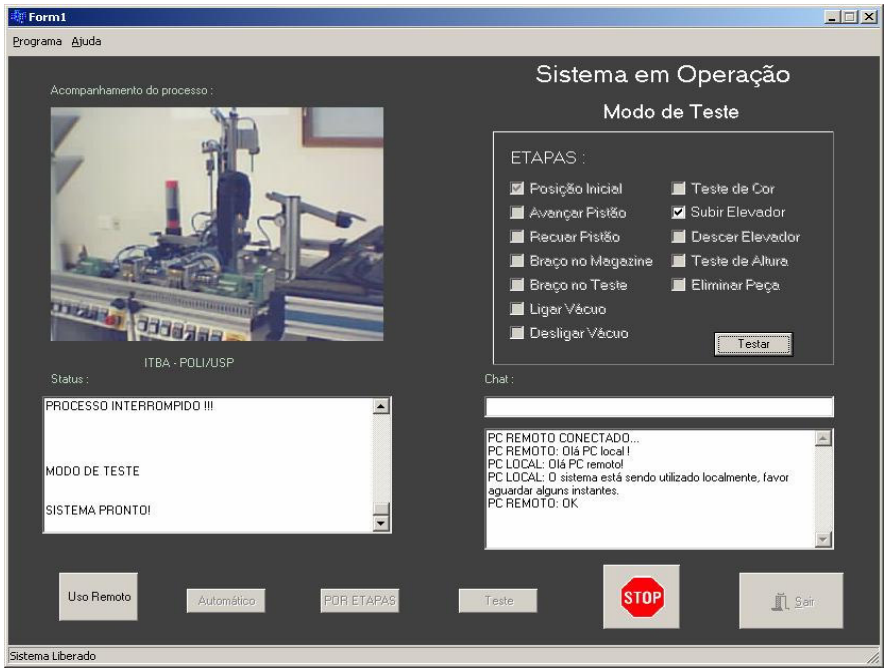
TELA DO PC LOCAL

Figura 41 – O usuário aciona no PC local a execução de uma tarefa no modo automático.



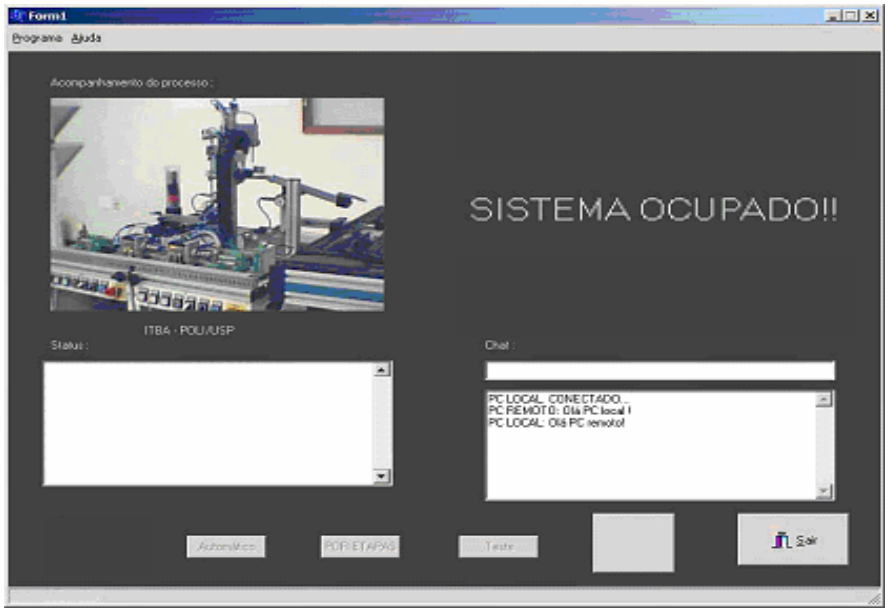
TELA DO PC LOCAL

Figura 42 – O usuário aciona no PC local a execução de uma tarefa no modo passo a passo.



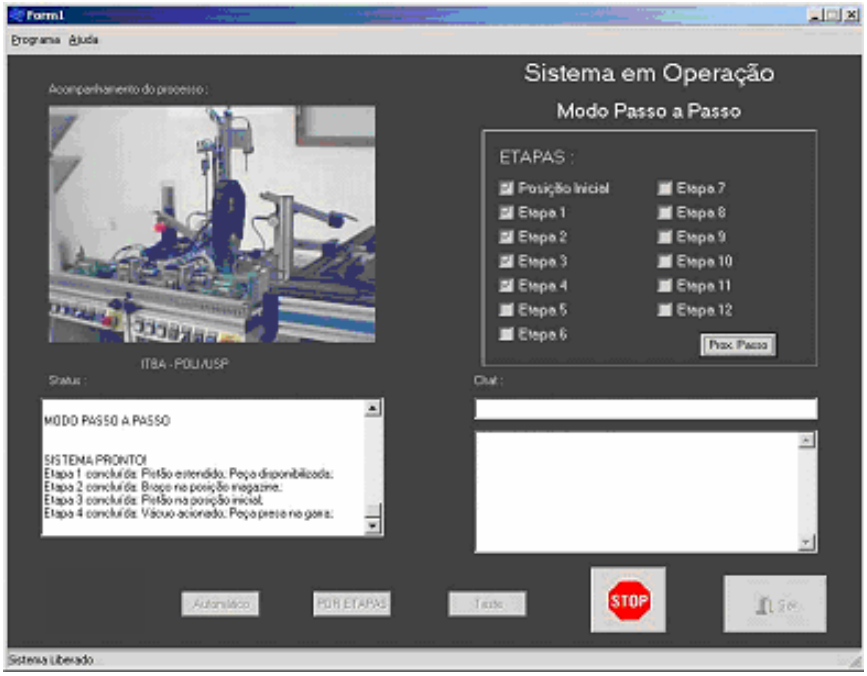
TELA DO PC LOCAL

Figura 43 – O usuário aciona no PC local a execução de um teste.



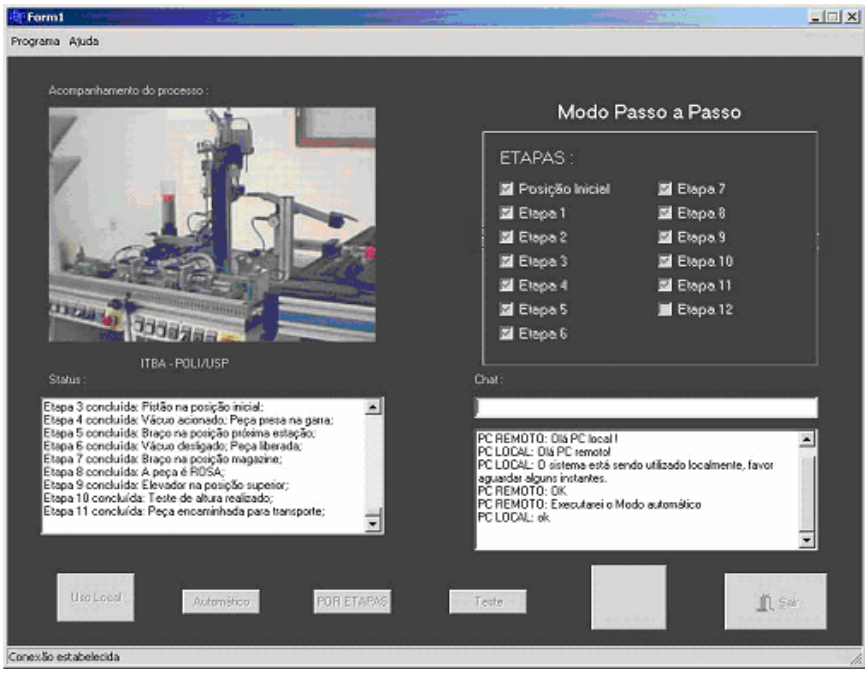
TELA DO PC REMOTO

Figura 44 – Tela no PC remoto enquanto o usuário local está no controle das estações de trabalho. O usuário remoto deve aguardar a permissão para ficar no controle, mas tem acesso ao c*hat*.



TELA DO PC REMOTO

Figura 45 – O usuário remoto agora com controle das estações de trabalho, aciona no PC remoto a execução de uma tarefa no modo passo a passo.



TELA DO PC LOCAL

Figura 46 – Tela no PC local enquanto o usuário remoto está no controle das estações de trabalho. O usuário local tem acesso ao c*hat*.

5.8. Sugestão de metodologia para especificação de comando e monitoramento remoto de sistemas produtivos

Com a experiência adquirida no desenvolvimento deste trabalho verificou-se que a especificação do comando e monitoramento de sistemas produtivos pode ser organizada por meio de uma seqüência de passos. Cada passo concerne um aspecto distinto do sistema e aumenta o nível de detalhe da especificação. Em cada um dos passos, uma série de questões deve ser respondida, analisando um aspecto determinado do sistema. O objetivo de organizar cada passo em um conjunto de perguntas é guiar e facilitar a aplicação da metodologia.

Passo 1: Controle ou monitoramento?

O primeiro passo consiste em definir os propósitos do acesso remoto ao sistema. O primeiro ponto a ser definido é se o sistema terá apenas funções de monitoramento ou se possuirá funcionalidades de comando. A resposta a esta pergunta variam de acordo com as limitações impostas pelas características do sistema e recursos disponíveis.

Em sistemas de acesso remoto, a informação adquirida do sistema de manufatura local é disponibilizada em tempo real ao destino remoto. Logo, o controle remoto é justificável apenas nos casos em que os dados recebidos são também processados e utilizados em tempo-real em seu destino. Quando a disponibilidade de dados em tempo real não é uma necessidade-chave, uma solução simples é armazenar os dados localmente e utilizar ferramentas padrão de compartilhamento de dados via internet.

As primeiras perguntas as serem respondidas são:

Pergunta 1.1: Quais são as vantagens em tornar as informações do sistema disponíveis para uso remoto em tempo-real?
Pergunta 1.2: Qual tipo de decisão pode ser tomada baseada nos dados disponibilizados no destino remoto?

As motivações para prover funcionalidades de controle podem ser investigadas através do estabelecimento de quais decisões são tomadas baseadas nos dados disponibilizados pelo acesso remoto. Basicamente, o ponto é determinar que módulos podem ser afetados pelas decisões do sistema remoto. Se as decisões tomadas interferem na evolução apenas do módulo remoto, o sistema é candidato a possuir somente monitoramento. Caso as decisões afetem a evolução do módulo local, então o sistema é candidato a comando e monitoramento remoto.

Passo 2: Especificação dos casos de uso e informações trocadas

Para determinar os dados a serem transmitidos entre os sistemas local e remoto, a primeira pergunta do passo 2 deve ser:

Pergunta 2.1: Quais são os casos de uso do sistema e quem são os atores?

Para responder a esta pergunta, o primeiro passo é desenvolver o diagrama UML de casos de uso do sistema. Em seguida, deve-se fazer uma lista dos dados necessários para tomar remotamente as decisões especificadas no passo 1. A segunda questão então é:

Pergunta 2.2: Quais informações são trocadas entre os sistemas local e remoto?

Passo 3: Análise de hardware

O passo 3 analisa a viabilidade do acesso remoto do ponto de vista do hardware:

Pergunta 3.1: Quais são os nós do sistema e como eles se comunicam entre si?

No caso de haver mais de um nó no sistema local, os nós locais podem se comunicar através de redes locais. Cada nó local pode se comunicar diretamente com o sistema remoto ou podem ser conectados a um servidor local que centraliza e gerencia a comunicação via Internet com o sistema remoto. Outro ponto é como os nós locais são conectados com bases de dados e registros históricos.

As próximas questões são:

Pergunta 3.2: Qual o hardware para a comunicação do sistema de manufatura local com a Internet?

Pergunta 3.3: Qual o tipo de linguagem de programação usada para desenvolver o sistema de acesso remoto e quais são as tecnologias disponíveis para implementar esta comunicação via Internet?

Esta última questão deve especificar se o sistema de acesso remoto será uma página de Internet ou um software desenvolvido para os propósitos em questão.

Passo 4: Refinamento de software

Neste passo, o caso de uso é detalhado. Isto pode ser feito através do desenvolvimento do Diagrama UML de Atividades. As perguntas do passo 4 são:

Pergunta 4.1: Qual é a seqüência de atividades para cada caso de uso?
Pergunta 4.2: O sistema de manufatura local pode ser acessado por múltiplos usuários remotos (simultaneamente ou não)?
Pergunta 4.3: No caso do sistema de controle remoto, como possíveis conflitos serão gerenciados?

Passo 5: Requerimentos para acesso remoto

O último passo da especificação do sistema remoto diz respeito aos requerimentos relacionados à natureza remota do controle e monitoramento do sistema. As perguntas a serem respondidas são:

Pergunta 5.1: O que acontece caso a comunicação falhe: o sistema local é capaz de detectar a falha de comunicação e deixar o sistema em um estado seguro?
Pergunta 5.2: Os atrasos de comunicação são críticos para a operação do sistema e o sistema local checa estes atrasos, tomando as ações pertinentes quando necessário?
Pergunta 5.3: Quais as facilidades disponíveis para o sistema remoto reagindo a falhas no sistema local de manufatura?
Pergunta 5.4: A natureza remota compromete a segurança do sistema?

6. Conclusões

As metas apontadas para a execução do projeto, ou seja, a elaboração do software para controlar e monitorar de forma integrada a estação de distribuição e a estação de testes e a definição de uma metodologia para especificação de comando e monitoramento remoto de sistemas produtivos foram devidamente atingidas.

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

- A implementação da transmissão de imagens entre os computadores não é uma tarefa trivial. Apesar de já se ter conhecimento dos recursos e funções envolvidas na sua implementação e de ser possível identificar diversas propostas/programas para este fim disponíveis na web, estes não têm código liberado a terceiros. Assim, a efetiva implementação da transmissão de imagens via Internet foi relativamente trabalhosa.
- A expansão do programa desenvolvido para a estação de trabalho também não é uma tarefa trivial. A depuração do programa resultante não é simples e muitas vezes, erros (às vezes triviais) são difíceis de serem identificados já que na etapa de compilação fica sempre a dúvida se o problema estava no código que já existia e foi alterado ou no que foi adicionado. Assim, o desenvolvimento levou mais tempo do que o previsto inicialmente.

Em relação ao presente projeto considera-se que apesar de se ter alcançado plenamente os objetivos previstos, existem ainda outras partes para que um sistema de manufatura como um todo possa ser telecomandado e monitorado remotamente, isto é, o sistema aqui desenvolvido para as estações de distribuição e de testes deve ser revisto para que soluções similares sejam implementadas para as outras estações de trabalho.

Quanto ao sistema específico desenvolvido para as estações de distribuição e de testes, alguns testes confirmaram o funcionamento esperado, mas, é necessária uma análise mais elaborada para avaliar aspectos como a confiabilidade do sistema e o grau de dependência das características da rede de comunicação.

Em relação à expansão para outras estações de trabalho está claro que o uso de redes de campo como o PROFIBUS deve ser considerado em futuras implementações, pois, é a solução prática que se tem disponibilizado nas plantas industriais.

Especialmente no caso deste projeto, ressaltam-se a seguir alguns pontos de aprendizado prático:

- Em se tratando de qualquer máquina, dispositivo ou software é fundamental dispor de uma “boa” documentação e uma “boa” estratégia para a leitura desse material (e/ou dos arquivos de apoio). Isso contribui efetivamente para a identificação tanto de

procedimentos recomendáveis como de novas abordagens para os problemas aparentemente não previstos. Por exemplo, no presente projeto, os arquivos de apoio do Borland C++ Builder foram de uma ajuda muito grande.

- Softwares que envolvem comunicação merecem cuidados especiais principalmente se houver mais de um tipo de comunicação deste software com o exterior. Isto é, a programação e funcionamento dependem de várias outras variáveis e assim cenários e situações específicas de teste e avaliação devem também ser considerados no desenvolvimento destes softwares. Por exemplo, no presente caso, tratar simultaneamente a comunicação serial concomitantemente com a comunicação via Internet (*sockets*) aumentou em muito o grau de dificuldade para o desenvolvimento do sistema.
- Sempre que se controla máquinas/dispositivos eletromecânicos por computador, nunca se deve esquecer que o tempo de execução do software no computador (ou no controlador) é muito menor que o tempo de execução de movimentos físicos.
- Trabalhar com transmissão de informação de vídeo pela Internet requer estudo do material sobre o assunto já disponível na Internet e dos arquivos de apoio de ferramentas tipo Borland C++ Builder. Realizar diversos programas de teste acompanhando o andamento do aprendizado em relação ao assunto é fundamental para assegurar o progresso do desenvolvimento dentro de um cronograma.
- Um "bom" planejamento de como programas relativamente complexos devem ser desenvolvidos diminui as chances de erros no meio do processo. Pois sem isso pode-se perder o foco do trabalho e a desorganização das idéias certamente prejudica o produto final e compromete o cronograma.
- Durante um projeto extenso, documentar cada etapa importante de trabalho cumprido é fundamental para a tarefa de documentação final do projeto.

7. Referências Bibliográficas

ALMEDA, W. M. Conhecendo o C++ Builder 6, 2003.

AP, www.apostilando.com.br, acessado em 18/11/2007.

BUENO, A. D. Apostila de programação orientada a Objeto em C++, v.0.4, 2002.

DBCB, www.dicasbcb.com.br, acessado em 22/09/2007, 10/10/2007, 27/10/2007.

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

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

GDH, www.guiadohardware.net, acessado em 05/09/2007, 07/09/2007.

LEE, W. B., LAU, H. C. W. Multi-agent modeling of dispersed manufacturing networks. Expert Systems with Applications, v.16, p.297-306, 1999.

MARTIN, V. Industrial Perspective on Research Needs and Opportunities in Manufacturing Assembly. Journal of Manufacturing Systems, v.14, n.1, 1995.

POHL, I. C++ para programadores de Pascal. Rio de Janeiro: Berkeley, 1991.

REMBOLD U.; NNAJI, 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, v.16, p.195-214, 1998.

SOUZA, L. E. Controladores Lógicos Programáveis. FUPAI, 2001.

CASSANDRAS, C.; LAFORTUNE, S. Introduction to Discrete Event Systems. Kluwer Academic Publ., 1999.

TRAVERSE, www.traverse.com/people/poinsett/bcbcomm.html, acessado em 14/10/2007.

VOTRE, V. P. C++ Explicado e Aplicado. Coleção ZeroErro em Engenharia de Software, v.2, 1998.

WARNOCK, I. G. Programmable Controllers: Operation and Application. New York.Prentice Hall, 1988.

XTRIMSOFT, http://www.xtrimsoft.com/downloads/BCBcomponent.htm, acessado em 22/09/2007.

Anexo A - Programação dos CP da Festo

A programação dos CPs da empresa Festo é executada no software FST V4.10, que é um ambiente que utiliza como linguagem de programação a lista de instruções.

A lista de instruções, ou STL (Statement List), é uma linguagem de programação de baixo nível baseada em texto. Esta permite ao programador descrever as etapas de funcionamento das funções do controlador a serem descritas por instruções relativamente simples. A estrutura da linguagem permite a manipulação eficiente de tarefas complexas.

A linguagem pode ser descrita como um conjunto de ramos ou etapas onde cada uma executa uma determinada rotina. Cada ramo é identificado como um STEP, e este pode conter um ou muitos blocos; um bloco completo é composto pela parte condicional e a parte de execução e apenas o primeiro bloco de um STEP pode ser incompleto, ou seja, conter apenas um comando de execução sem que lhe seja imposta qualquer condição.

A passagem aos passos subseqüentes é gradual, embora seja possível executá-los em qualquer ordem desejada através do comando JMP TO, cuja função é remeter à ramificação desejada. O programa só avança para o próximo passo se no último bloco do passo corrente um comando THEN ou OTHRW (que indicam a parte de execução do código) foi executado.

Um exemplo simples de um programa nesta linguagem pode ser visto na Listagem 1:

```
STEP label 1
        IF                              I1.0
        THEN    SET                     F1.5
        OTHRW   RESET                   F1.5

STEP label 2
        THEN    RESET                   F0.0
        IF                              F1.5
        THEN    SET                     O0.7
                SET                     F0.0
        OTHRW SET                       O0.0
         JMP TO label 1

STEP label 3
        IF                              F0.0
        AND                             I0.0
        THEN    SET                     O0.4

STEP label 4
...
```

Listagem 1 – Trecho de Programa em Lista de Instruções

Organização do Programa

Na criação de um programa por lista de instruções, este é composto, além dos programas locais, por módulos, conhecidos por CFM (*Calling Function Module*) e CMP (*Calling Program Module*), que serão sucintamente explicados a seguir:

- CFM (*Calling Function Module*): quando o comando CFM é utilizado, um módulo de função é chamado. Um módulo de função é uma sub-rotina que pode ser criada em C ou na própria linguagem de lista de instruções e não apresenta a necessidade de ser organizada em passos. Quando o comando é utilizado, não ocorrem mudanças na tarefa executada, mas parâmetros são transferidos para o programa local em execução;

- CMP (*Calling Program Module*): quando o comando CMP é utilizado, um módulo de programa é chamado. Um módulo de programa também é uma sub-rotina que pode ser criada em C ou na própria linguagem de lista de instruções e não apresenta a necessidade de ser organizada em passos. A diferença deste para o CFM consiste no fato de que, quando executado o CMP, ocorre necessariamente mudança na tarefa executada.

Principais Comandos

A fim de compreender a lógica do código apresentado, discorre-se abaixo sobre os principais comandos desta linguagem:

- Comando IF: O comando IF determina a declaração de condições. Com a introdução deste comando no código, os operandos podem ser chamados e arranjados de maneira a formar expressões lógicas e aritméticas, que representarão condições para a execução de processos posteriores.

  Exemplo:

```
IF            I1.0    " IF 1 signal to I1.0
AND    N      I1.1    " AND 0 signal to I1.1
...
```

- Comando THEN: o comando THEN inicia a execução de um processo. Tal processo será executado se as possíveis condições anteriores estiverem satisfeitas.

  Exemplo:

```
THEN   LOAD           V100
           TO            TP7
...
```

- Comando OTHRW: inicia uma segunda execução alternativa de um processo. Este processo só será executado se as condições propostas para a execução de uma

determinada tarefa não forem satisfeitas (ou seja, se o processo determinado pelo comando THEN não puder ser executado).

Exemplo:

```
 ...
THEN    SET                    O1.0
OTHRW   RESET                  O1.0
```

- Comando SET: ajusta o valor do operando indicado para verdadeiro (1).
- Comando RESET: ajusta o valor do operando indicado para falso (0).
- Comando LOAD: Carrega um valor (de um ou mais bits), que através do comando TO é transferido para um operando.

Exemplo:

```
THEN    LOAD    I0.1    " Single-bit
              TO      F0.1    " Single-bit
      LOAD    V500    " Multi-bit
              TO      TP31    " Multi-bit
```