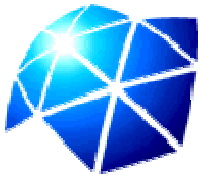
UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
CAMPUS DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA



Manipulação Remota de um Braço Mecânico (SCORBOT ER - III) utilizando a Rede Mundial de Computadores

Marcos Antonio Estremote

Dissertação submetida à Universidade Estadual Paulista – UNESP, Campus de Ilha Solteira, para preenchimento dos requisitos necessários para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Nobuo Oki

ILHA SOLTEIRA - SP, JANEIRO DE 2006

Manipulação Remota de um Braço Mecânico (SCORBOT ER - III) utilizando a Rede Mundial de Computadores

MARCOS ANTONIO ESTREMOTE

DISSERTAÇÃO SUBMETIDA À UNIVERSIDADE ESTADUAL PAULISTA – UNESP, CAMPUS DE ILHA SOLTEIRA, PARA PREENCHIMENTO DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA ELÉTRICA.

COMISSÃO EXAMINADORA:

Prof. Dr. NOBUO OKI

DEPARTAMENTO DE ENGENHARIA ELÉTRICA – FE – UNESP – CAMPUS DE ILHA SOLTEIRA /SP

Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA – FE – UNESP – CAMPUS DE ILHA SOLTEIRA /SP

Prof. Dr. JOSÉ RAIMUNDO DE OLIVEIRA

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL – FEEC – UNICAMP – CAMPINAS/SP

ILHA SOLTEIRA - SP, JANEIRO DE 2006

À minha família, sobretudo meus pais:

Lenir Almeida Estremote

João Antonio Estremote "In Memorian",

pela educação pautada, pela honestidade

e trabalho, meu profundo reconhecimento e

consideração.

Às minhas filhas Izabella Prato Estremote e

Maria Eduarda Prato Estremote e milha

esposa Elaine Prato Estremote, pontos de luz,

que a cada dia me fortalece e edifica para a

conquista dos meus sonhos.

Ofereço!

"Porque ficar de braços cruzados se o maior homem do mundo morreu de braços abertos. Pois vitorioso não é aquele que vence o outro, mas sim a si mesmo".

AGRADECIMENTOS

A minha gratidão, primeira e principalmente, é a Deus, forte rochedo em que me abrigo autor e guia de nossas vidas, por tanto amor e tanto cuidado a mim dispensados.

Agradeço a meus pais, minha esposa, aos meus irmãos (Mário Márcio Estremote e Marcelo Estremote) e familiares, que sempre lutaram por seus ideais me ensinando, com isso, a persistir na caminhada, por mais árdua que ela seja.

Ao professor Doutor Nobuo Oki, que com sua competência e habilidade, unido à sua receptividade, amizade e confiança se fez presente em toda esta caminhada.

Aos amigos Edilson Souza dos Santos, Vinícius de Araújo Maeda, Horêncio Serrou Camy Filho, Odacir de Almeida Neves, Tércio Alberto dos Santos Filho e Cristiano Pires Martins que os considero um exemplo a ser seguido tanto no campo profissional como pessoal.

A todos os colegas de trabalho que direta ou indiretamente, cooperaram para a realização deste trabalho.

"O importante não é estar aqui ou ali, mas ser, e ser é uma ciência delicada feita de pequenas e grandes observações do cotidiano, senão executarmos essas observações não chegamos a ser, apenas estamos e desaparecemos".

EPÍGRAFE

“A ARTE DE VIVER É UMA TÁTICA EM QUE POR MUITO TEMPO TEREMOS DE SER APRENDIZES”.

RESUMO

Neste trabalho descreve-se um Software de comando para o acionamento de um braço mecânico do robô SCORBOT ER - III. O software desenvolvido tem a capacidade de controlar e monitorar o robô remotamente em tempo real através da Rede Mundial de Computadores (WWW), utilizando bibliotecas de JAVA como: métodos nativos (JNI - JAVA Native Interface), Invocação de Métodos Remotos (RMI – Remote Method Invocation), Conectividade com Banco de Dados JAVA (JDBC – JAVA Database Connecvity) e JMF (JAVA Media Frameworks) com o Protocolo de Tempo Real – RTP (Real Time Protocol). Para controle do robô, o circuito de controle originalmente desenvolvido pelo fabricante, foi reprojetado utilizando-se o ambiente Max+Plus II da Altera e a conexão entre Robô e o PC é feita através de um dispositivo lógico programável tipo FPGA, que recebe os comandos provenientes da Porta Paralela do PC, o monitoramento através de câmeras digitais do tipo WEBCAM conectadas em uma Porta do tipo USB.

Palavras-chave: Controle Remoto, Robô SCORBOT ER – III, Linguagem JAVA, Tempo Real.

ABSTRACT

This work describes the development of a Software for the control of a mechanical arm type SCORBOT ER - III. This software has the capacity to control and to monitor the robot remotely in real time through the World Wide Web (WWW), using libraries of JAVA as: native methods (JNI-JAVA Native Interface), Invocation of Remote Methods (RMI - Remote Method Invocation), Connecvity with database JAVA (JDBC - JAVA Database Connecvity) and JMF (JAVA Media Frameworks) with the Protocol of Real Time - RTP (Real Time Protocol). The robot control circuit was redesigned using the Altera Max+Plus II environment and the connection between the robot and personal computer was made by the Parallel Port and digital cameras of the type WEBCAM, connected USB port.

Keywords: Remote control, Robot SCORBOT III - ER, Language JAVA, Real Time.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interfaces* (Interfaces de Programação de Aplicativos)
CGI	*Common Gateway Interface* (Interface de Gateway Comum)
CI	*Circuito Integrado*
DEE	*Departamento de Engenharia Elétrica*
DLL	*Dynamic Link Libraries* (Bilbioteca Dinâmica)
FEIS	*Faculdade de Engenharia de Ilha Solteira*
FPGA	*Field Programmable Gate Arrays* (Matriz de Portas Programáveis em Campo)
FTP	*File Transfer Protocol* (Protocolo de Transferência de Arquivos)
HTTP	*HyperText Transfer Protocol* (Protocolo de Transferência de Hipertexto)
IEEE	*Institute of Electrical and Electronics Engineers* (Instituto de Engenheiros Elétricos e Eletrônicos)
I/O	*Input/Output* (Dispositivo de Entrada e Saída)
JDBC	*JAVA Database Connectivity* (Conectividade com Banco de Dados JAVA)
JIT	*Just-in-Time* (Tempo de Execução)
JMF	*JAVA Media Frameworks*
JNI	*JAVA Native Interface* (Interface Nativa JAVA)
JVM	*JAVA Virtual Machine* (Máquina Virtual JAVA)
Kbps	Kilobits/segundo
LAN	*Local Area NetWork* (Rede de Área Local)
Mbps	Megabits/segundo
PC	*Personal Computer* (Computador Pessoal)

RMI	*Remote Methods Invocation* (Invocação de Métodos Remotos)
RTCP	*Real Time Control Protocol* (Protocolo de Controle de Tempo Real)
RTP	*Real Time Protocol* (Protocolo de Tempo Real)
SQL	*Structured Query Language* (Linguagem de Consulta Estruturada)
TCP/IP	*Transmission Control Protocol/Internet Protocol* (Protocolo de Controle de Transmissão/Internet Protocol)
UDP	*User Protocol Datagram* (Protocolo Datagrama de Usuário)
UML	*Unified Modeling Language* (Linguagem Unificada de Modelagem)
URL	*Uniform Resource Location* (Localização de Recursos Uniformes)
USB	*Universal Serial Bus* (Barramento Serial Universal)
VHDL	*Very High Speed Integrated Circuit Hardware Description Language* (Linguagem de Descrição de Hardware para Circuitos Integrados de alta velocidade)
WWW	*World Wide Web*

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

1.1 – INTRODUÇÃO

Um dos tópicos fascinantes de engenharia é a área de robótica[1]. Os robôs desempenham tarefas repetitivas e aquelas que colocariam os operários em condições insalubres ou de risco, sendo seu uso fundamental nas atividades industriais, principalmente na indústria automobilística onde precisão e qualidade são imprescindíveis. Definem-se como robôs, máquinas capazes de executar determinadas tarefas de forma programada. No projeto de desenvolvimento de robôs estão envolvidos conhecimentos multidisciplinares, tais como: mecânica, eletrônica (analógica, digital e potência), controle e entre outras.

De acordo com a literatura [1], o primeiro trabalho envolvendo a manipulação remota de robôs via *World Wide Web*[2] (WWW) foi efetuado no Projeto denominado Mercúrio coordenado por Ken Goldberg e Michel Mascha da University Southern Califórnia. Atualmente, existem diversos grupos trabalhando com este conceito com finalidades que variam desde a educacional, visando despertar o interesse de alunos para áreas de computação, até o uso de sofisticadas teorias de controle, tais como redes neurais e técnicas de identificação para orientação de robôs em áreas remotas (manutenção de satélites, por exemplo) ou de risco (rastreamento de bombas e minas, prospecção e montagem de equipamentos para extração de petróleo em águas profundas).

Por outro lado, o uso da rede mundial de computadores, cada vez mais popular, atinge um público bastante heterogêneo, entre os quais, alunos que possam se interessar pela área de tecnologia. O desenvolvimento de projetos, no qual se torna possível à interação de usuário com robô, é grande oportunidade para despertar este interesse. A rede mundial de

[1] É o conjunto dos estudos e das técnicas que permitem a utilização de robôs na automação.
[2] Rede Mundial de Computadores.

computadores possibilita, pela sua concepção, o acesso quase universal dos trabalhos desenvolvidos, pois a plataforma de uso encontra-se já definida e disponível. Na rede, existe uma série de páginas que efetuam este trabalho, no entanto o país carece de iniciativas neste sentido.

Na área industrial os robôs já fazem parte das plantas industriais onde os requisitos de alta produtividade e confiabilidade são necessários.

Para o público em geral, a área de robótica desperta fascínio pela divulgação de recentes conquistas obtidas, tais como o envio de sondas tele-operadas a Marte, as aplicações na indústria automobilística, sua utilização em brinquedos e mesmo sua utilização em filmes de ficção.

1.2 – CONTEXTUALIZAÇÃO DO PROJETO

O projeto para o acionamento de um robô remotamente envolve a aplicação de diversas técnicas e metodologias em relação ao interfaceamento e comunicação. O sistema original não disponibilizava condições de novas atualizações devido a sua estrutura fechada, os circuitos de acionamento e controle tornaram-se obsoletos, além do software de comando. Com isso, surgiu a necessidade de construir um novo hardware e software, flexível e aberto.

1.2.1 – OBJETIVOS DO TRABALHO

Neste trabalho descreve-se o desenvolvimento de um software portável[3] que permite o acionamento remoto de um robô, através da rede mundial de computadores como plataforma de controle. O sistema original de acionamento foi reestruturado e é controlado atualmente por um microcomputador tipo PC utilizando a porta paralela para controle do braço. Os aspectos enfocados neste estudo são direcionados ao desenvolvimento de um software

[3] Aplicações capazes de serem executadas em uma variedade de sistemas operacionais e de arquiteturas de hardware.

responsável pela manipulação do braço mecânico do SCORBOT ER-III via WWW, análise e definição da linguagem JAVA utilizada no desenvolvimento, incluindo os protocolos de comunicação até a elaboração do ambiente de interface usuário-máquina.

Em um trabalho existente [2] indicou-se a possibilidade de utilização de linguagem JAVA para o acionamento e controle de dispositivos. Além de ser uma linguagem de programação[4], pode também ser utilizado como plataforma computacional[5] [3], que apresenta várias características tais como ser orientada a objetos, ser robusta, portável e segura. Alguns ambientes necessários para a elaboração deste trabalho estão descritos na literatura [4], [5] e [6].

A característica de independência de arquitetura do JAVA, o código fonte, depois de compilado pode ser executado em muitos processadores, é o grande motivo pelo quais seus programas são portáveis. Outro aspecto da portabilidade envolve a estrutura ou os tipos de dados inerentes da linguagem, como inteiro, *string* e ponto flutuante.

JAVA tira proveito dos padrões IEEE para os tipos de dados comuns em várias arquiteturas de computadores. Por exemplo, em JAVA uma estrutura de dados *float* sempre estará de acordo com o padrão IEEE 754 para números de ponto flutuante, enquanto o tipo de dado *int* será sempre um inteiro de 32 bits com complemento de dois sinalizado. Além disso, resolve questões relacionadas com dependências de ordem de bytes de *hardware* com terminações *big endian*[6] e *little endian*[7]. Estes aspectos o diferem de outras linguagens, como o C++, uma vez que, o binário de código de *bytes* não possui dependência de implementação, portanto é portável para várias plataformas diferentes. É com base nestes conceitos que optou-

[4] Conjunto de instruções e regras de composição e encadeamento, por meio do qual se expressam ações executáveis por um computador, seja diretamente, seja por meio de processos de compilação, interpretação ou montagem.
[5] Arquitetura ou padrão de um tipo de computador, ou de sistema operacional.
[6] O número hexadecimal A02B seria armazenado como A02B pelo método *big endian.* O método *big endian* é utilizado pelos microprocessadores Motorola.
[7] O número hexadecimal A02B seria armazenado como 2BA0 pelo método *little endian.* Os microprocessadores Intel fazem uso do método *little endian.*

se em utilizar JAVA para o desenvolvimento do software de controle na manipulação do braço mecânico, tornando o aplicativo final disponível para diversas plataformas.

1.2.2 – MOTIVAÇÃO

A utilização de robôs na automação industrial e em locais de difícil acesso por seres humanos ou em situações de alto risco cresce rapidamente em âmbito mundial. Da mesma forma, provavelmente a exploração espacial também dependerá durante longo tempo de robôs, capazes de atuar de modo satisfatório em ambientes desconhecidos de outros planetas. Os países que não dominarem o ciclo de projeto, construção e operação de robôs autônomos estarão comprometendo fortemente sua capacidade de competição em um mercado globalizado, uma vez que, tais equipamentos já são considerados essenciais em vários tipos de atividades industriais. Na indústria automobilística, por exemplo, utilizam de forma rotineira robôs de soldagem, com os quais é possível a construção de complexas estruturas dos modernos automóveis com baixo custo e altíssima qualidade.

Robôs autônomos demandam um conjunto de tecnologias, com a participação de diversas especialidades. Um robô autônomo deve ser dotado de visão computacional e/ou sensores, capazes de posicioná-lo em um ambiente desconhecido, associado a um programa de controle capaz de atuar em situações de conflito. Adicionando-se o requisito de ação cooperativa entre máquinas inteligentes, leva-se ainda à necessidade de se programar sistemas de comunicação entre os robôs e a divisão de tarefas.

O estudo da robótica autônoma multi-agente tem sido realizada com baixa intensidade no Brasil. Entre os fatores limitantes na atuação dos pesquisadores, destaca-se, o alto custo de montagem de um laboratório de robótica. Por exemplo, para adquirir e manter um robô utilizado em atividades de pesquisa em diversos países é necessário um grande investimento, tornando difícil tal aquisição no contexto nacional.

Por outro lado, os jovens estudantes, ao fazerem sua escolha de um curso universitário, parece se afastarem cada vez mais das carreiras tecnológicas. Em parte trata-se de uma inclinação natural individual, mas é também evidente que muitos jovens vêem na aridez da Matemática e da Física ensinada nos colégios um fator desmotivador para optarem por cursos superiores de Engenharia, Física, Matemática e outras ciências consideradas difíceis. Na verdade o que ocorre é um desconhecimento total da aplicabilidade prática dos conceitos teóricos da Física e da Matemática às situações cotidianas, como se existisse a teoria dissociada do mundo real. Para reverter esse processo, é preciso mostrar aos jovens que as fórmulas e equações, descrevem fenômenos físicos reais, como a trajetória de um braço mecânico. Ou seja, a proposição deste trabalho, nada mais é, do que uma aplicação prática e real, de ferramentas físicas, matemáticas e computacionais avançadas.

1.2.3 – JUSTIFICATIVA

O Departamento de Engenharia Elétrica (DEE – FEIS UNESP) obteve o empréstimo de um braço mecânico SCORBOT ER - III de cinco graus de liberdade, junto ao Departamento de Engenharia Civil da mesma Instituição. No entanto, os circuitos de acionamento, controle e o software de controle tornaram-se desatualizados, passando por mais de quinze anos sem atualização devido a sua estrutura fechada tipo caixa preta. Assim, fez se necessário a construção de um novo sistema aberto de hardware (circuito de controle) e software foi de essencial importância para que se possa atualizá-los constantemente. O circuito de controle do SCORBOT ER - III foi desenvolvido por um aluno de graduação da Faculdade de Engenharia de Ilha Solteira em seu Trabalho de Conclusão de Curso [7]. Com a utilização da Linguagem JAVA para a implementação do software de controle, o braço pode ser controlado através da Internet por uma aplicação cliente/servidor, fazer a interligação (*link*) entre o campus de Bauru, Guaratinguetá e Ilha Solteira.

1.3 – ORGANIZAÇÃO DO TEXTO

No segundo capítulo é conceituado o robô que serviu de base para a criação do software de manipulação. A linguagem de programação JAVA como ferramenta de programação, é apresentada no terceiro capítulo. No quarto, é feita a descrição da implementação do software de controle para a manipulação do braço utilizando a linguagem proposta. No quinto capítulo são mostrados os diagramas de classes do software de controle desenvolvido. No sexto capítulo, são apresentados os testes de comunicação do software e por fim, no sétimo capítulo são relatadas as conclusões do trabalho.

CAPÍTULO 2 - BRAÇO MECÂNICO SCORBOT ER - III

Neste capítulo destacam-se as definições de robótica, abordando sua história e por fim as características do robô estudado neste projeto de Mestrado.

2.1 – INTRODUÇÃO

Atualmente torna-se cada vez mais necessário a realização de tarefas com eficiência e precisão. Além disto, existem atividades a serem realizadas em lugares onde a presença humana torna-se difícil, arriscadas e até mesmo impossíveis, como o fundo do mar ou a imensidão do espaço, ou mesmo locais na indústria onde o risco de acidentes com perda dos membros é muito alto.

Para realizar essas tarefas, se faz necessária à presença de dispositivos robóticos, que as realizem, sem colocar em risco a vida do homem.

A Robótica é uma ciência multidisciplinar, altamente ativa, que busca o desenvolvimento e a integração de técnicas e algoritmos para a criação de robôs. Esta ciência envolve áreas como engenharia mecânica, engenharia elétrica, inteligência artificial, *design*, entre outras, com perfeita harmonia.

2.2 – ROBÓTICA E SUA HISTÓRIA

A terminologia robô origina-se da palavra tcheca "*robotnik*", ao qual tem o significado de servo, este termo foi idealizado primeiramente por Karel Capek em 1923, nesta época a idéia de um "homem mecânico" parecia vir de alguma obra de ficção [8].

O desejo de se idealizar os robôs não pertence apenas ao homem moderno, existem alguns acontecimentos históricos que já apontavam para esta idéia, por exemplo, as diversas referências sobre o "Homem Mecânico" construído por relojoeiros com a finalidade de se

exibir em feiras; as "Animações Mecânicas" como o leão animado de Leonardo da Vinci, e seus esforços para fazer máquinas que reproduzissem o vôo das aves [8]. Contudo, certamente estes dispositivos eram muito limitados, pois não podiam realizar mais do que uma tarefa, ou um número reduzido delas.

A grande evolução do desenvolvimento da robótica foi no início do século XX, quando houve uma grande necessidade do aumento na produtividade, porém sem perder a qualidade dos produtos. Foi nesta época que o robô industrial encontrou suas primeiras aplicações, sendo George Devol considerado o pai da robótica industrial [8].

Devido aos inúmeros recursos que os hardwares e softwares oferecem e com o avanço da inteligência artificial, a robótica atravessa uma época de contínuo crescimento que permitirá, em um curto espaço de tempo, o desenvolvimento de robôs inteligentes.

2.3 – CLASSIFICAÇÃO DOS ROBÔS

A classificação dos robôs se dá basicamente de acordo com as aplicações e maneiras de executar as ações [9]:

1. Robôs Inteligentes: estes robôs são acionados por sistemas multifuncionais controlados por computador onde, são capazes de interagir com seu ambiente através de sensores e de tomar decisões em tempo real.

2. Robôs com controle por computador: já este tipo de robô, apesar de parecidos com os robôs inteligentes, não possuem a capacidade de interagir com o ambiente. Quando estes robôs são equipados com sensores e software adequado, passam a ser robôs inteligentes.

3. Robôs de aprendizagem: os robôs de aprendizagem, apenas são capazes de repetir uma seqüência de movimentos, realizados por meio de um operador ou quando memorizados.

4. Manipuladores: os manipuladores são considerados sistemas mecânicos multifuncionais, onde o sistema de controle possibilita administrar o movimento de seus membros das seguintes formas:

4.1. manual: quando o operador controla diretamente os movimentos;

4.2. seqüência variável: quando é possível alterar algumas das características do ciclo de trabalho.

Quanto à classificação dos robôs em relação ao controle de seus movimentos, têm-se as seguintes configurações:

1. Sem controle-servo: o controle de movimento em um robô sem controle-servo é feito por um programa que controla o movimento dos diferentes componentes do robô, esta movimentação se realiza em um posicionamento “ponto-a-ponto" no espaço;

2. Com controle-servo: este tipo de controle permite duas formas de trabalho:

2.1. controle dos movimentos dos membros do robô em função de seus eixos. Os movimentos podem ser realizados “ponto-a-ponto” ou com trajetória contínua;

2.2. os movimentos se estabelecem da respectiva posição de seus eixos de coordenada e da orientação da mão ferramenta do robô.

Após esta breve explanação com relação aos conceitos básicos sobre robótica, observa-se no item seguinte as características do robô SCORBOT ER - III, os quais foram essenciais para o desenvolvimento deste projeto de pesquisa.

2.4 – SCORBOT ER - III

Para o desenvolvimento e testes do software de controle, foi utilizado um braço mecânico educacional SCORBOT ER – III, fabricado pela empresa ESHED ROBOTEC,

situada na cidade de Tel-Aviv - Israel, este robô está disponível no Departamento de Engenharia Elétrica da Faculdade de Engenharia, Campus de Ilha Solteira – UNESP. Este braço tem cinco graus de liberdade, controlado por oito motores de corrente contínua, possui juntas de rotação, caracterizando-o como um robô com articulação vertical. O mecanismo é acionado por *driver* elétrico, atua com garra tipo “dois dedos” e seu comando é feito ponto-a-ponto.

O circuito de controle original do braço mecânico foi reestruturado para que pudesse haver a comunicação entre o robô e um computador pessoal.

As figuras 2.1 e 2.2 mostram algumas fotos do braço mecânico SCORBOT ER - III.

Figura 2.1: Braço Mecânico SCORBOT ER – III

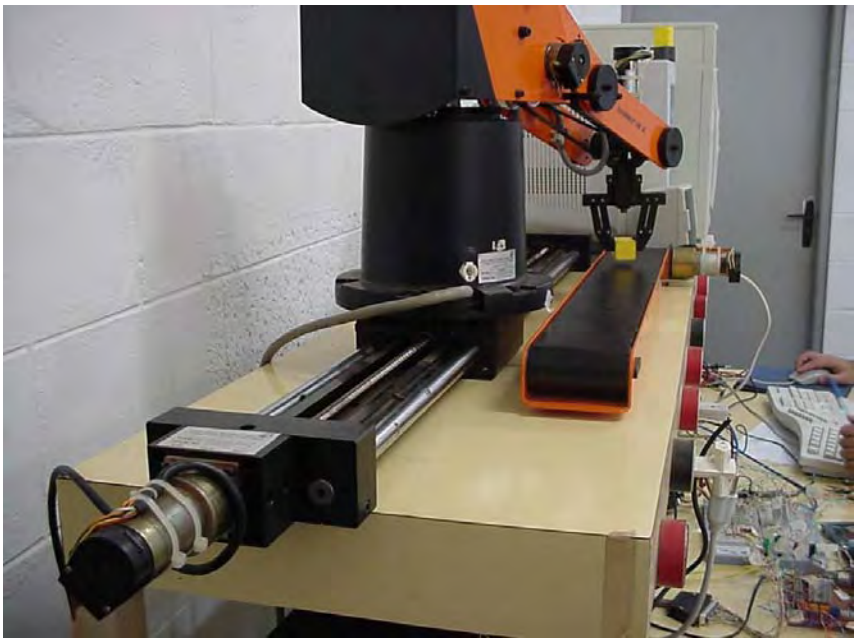


Figura 2.2: Foto do braço onde se pode visualizar a esteira e sua base deslizante pertencentes ao robô

Os robôs podem executar movimentos no espaço, transferindo objetos e ferramentas de um ponto para outro, instruído por um controlador e informando sua posição no espaço através de sensores. Nas extremidades existem atuadores que realizam a execução das tarefas. Todo robô pode ser composto de uma série de vínculos e juntas, onde a junta conecta dois vínculos permitindo o movimento relativo entre eles, como mostra a figura 2.3. Em geral, robôs possuem uma base fixa e o primeiro vínculo está preso a esta base. A mobilidade dos robôs depende do número de vínculos e articulações.

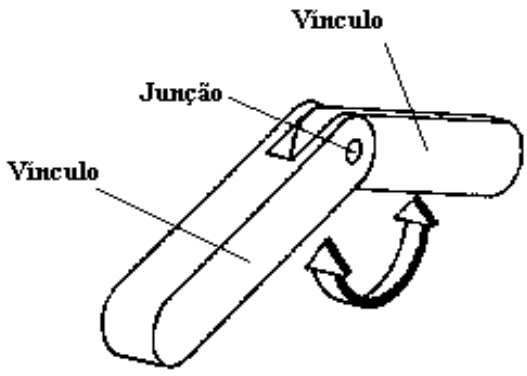


Figura 2.3: Junta e vínculos em um braço de robô

2.5 – GRAUS DE LIBERDADE

O número de articulações em um braço do robô é também referenciado como grau de liberdade. Quando o movimento relativo ocorre em um único eixo, a articulação possui um grau de liberdade e quando ocorrer em mais de um eixo, terá dois graus de liberdade. A maioria dos robôs tem entre 4 a 6 graus de liberdade. Já o homem, do ombro até o pulso, tem 7 graus de liberdade [7].

2.6 – FORMA DE ACIONAMENTO (*DRIVER ELÉTRICO)*

O *driver* elétrico aciona motores elétricos que podem ser de corrente contínua, de passo ou corrente alternada. Muitos robôs novos têm *drivers* para motor corrente contínua devido a sua boa precisão e simplicidade no controle do motor elétrico [7].

As vantagens do *driver* elétrico são:

- Eficiência calculada, controle preciso;
- Envolve uma estrutura simples e de fácil manutenção;
- Não requer uma fonte de energia cara;
- Custo relativamente pequeno.

As desvantagens são:

- Não pode manter um momento constante nas mudanças de velocidade de rotação;
- Sujeitos a danos para cargas pesadas, suficientes para parar o motor;
- Baixa razão de potência de saída do motor e seu peso, necessitando um motor grande no braço.

2.7 – CLASSIFICAÇÃO POR TIPO DE JUNTA (*JUNTAS DE ROTAÇÃO*)

Esta conexão permite movimentos de rotação entre dois vínculos. Os vínculos são unidos por uma dobradiça comum, com uma parte podendo se mover num movimento cadenciado em relação à outra, [7] como mostrado na figura 2.4. As juntas de rotação são utilizadas em muitas ferramentas e dispositivos, tais como tesouras, limpadores de pára-brisa e quebra-nozes [7].

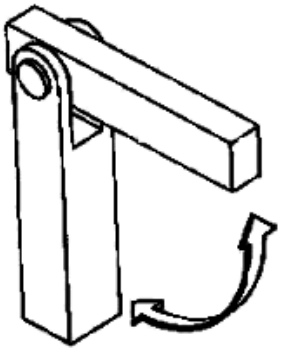


Figura 2.4: Junta de rotação.

2.8 – ARTICULAÇÃO VERTICAL

É usual classificar os robôs de acordo com o tipo de junta, ou mais exatamente, pelas três juntas mais próximas da base do robô. Essa divisão em classes fornece informações sobre características dos robôs em várias categorias importantes [7];

1. Espaço de trabalho;
2. Grau de rigidez;
3. Extensão de controle sobre o curso do movimento;
4. Aplicações adequadas ou inadequadas para cada tipo de robô.

Os robôs de articulação vertical caracterizam-se por possuir três juntas de revolução, como mostrado na figura 2.5.

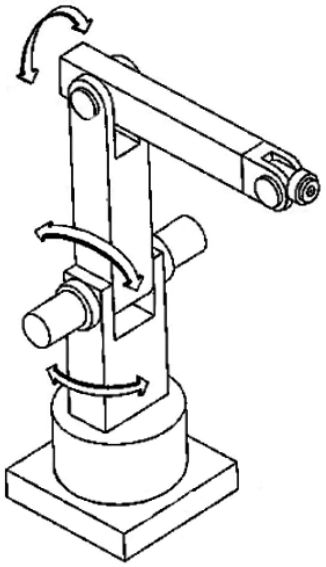


Figura 2.5: Robô com Articulação Vertical

Sua área de atuação é maior que qualquer tipo de robô, tendo uma baixa rigidez mecânica. Seu controle é, complicado e difícil, devido às três juntas de revolução e devido a variações no momento de carga e de inércia [7].

2.9 – ATUADOR DO SCORBOT ER - III

O atuador (*end effector*) é todo um sistema montado na extremidade do vínculo mais distante da base do robô, cuja tarefa é agarrar objetos, ferramentas ou transferi-las de um lugar para outro. São exemplos de atuadores: a pistola de solda, garras e pulverizadores de tintas. A operação do atuador é o objetivo final na operação de um robô, assim todos os demais sistemas (unidades *driver*s, controles, etc.) são projetados para habilitar sua operação.

O atuador é de extrema importância na execução de uma tarefa, portanto é necessário que o mesmo seja adequadamente projetado e adaptado as condições do seu meio e área de

trabalho. Existem vários tipos de atuadores, mas comentar-se-á somente dos atuadores do tipo garra, pois é, componente pertencente ao robô estudado em questão.

Uma garra é comparável à mão humana. No entanto, ela não é capaz de simular seus movimentos, resultando na limitação dos movimentos a uma faixa de operações. A grande demanda tem levado ao desenvolvimento de garras que podem manusear objetos de diferentes tamanhos, formas e materiais. O robô SCORBOT ER - III possui garra do tipo dois dedos. Essa garra é o tipo mais comum e com grande variedade. São diferenciadas umas das outras pelo tamanho ou movimento dos dedos. Na figura 2.6 ilustra-se o movimento paralelo de uma garra de dois dedos e a figura 2.7 o atuador do robô SCORBOT ER - III.

A principal desvantagem desta garra é a limitação da abertura dos seus dedos restringindo a sua operação em objetos cujo tamanho não exceda esta abertura.

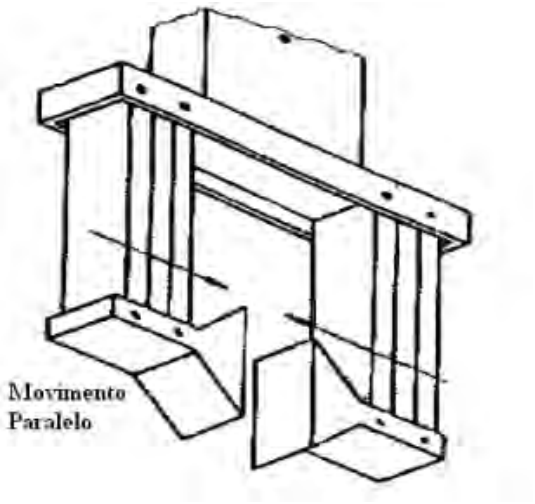


Figura 2.6: Garra de movimento paralelo.

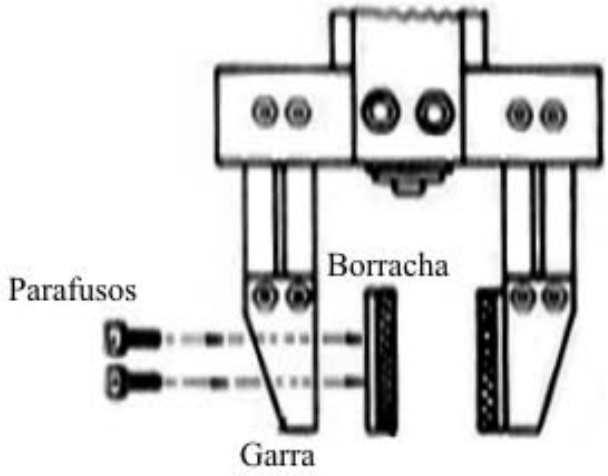


Figura 2.7: Garra do SCORBOT ER - III

Diante destes fatos verifica-se que o desenvolvimento e produção de garras é um estágio importante no projeto de robôs para tarefas particulares. Normalmente, os fabricantes vendem robôs sem o atuador. As garras e as ferramentas são escolhidas e adaptadas pela equipe de engenharia que instala o robô no local de trabalho.

2.10 – CONTROLE DE TRAJETÓRIA

Cada tarefa executada por um robô pode ser considerada como uma série de operações, através da qual o atuador é movido pelo braço do robô entre pontos pré-determinados e operado como programado. O controle de trajetória pode ser classificado em dois métodos: ponto-a-ponto e contínuo [7]. O controle de trajetória ponto-a-ponto é uma das características do SCORBOT.

Antes da descrição do método de controle ponto-a-ponto, é definido alguns termos:

- Ponto: localização no espaço em direção ou através do qual o atuador é movido por uma operação do braço do robô;
- Passo: uma parte do programa operacional do robô. A cada passo, o robô executa uma atividade;
- Série: uma coleção de passos que combinados formam o programa operacional do robô.

No controle ponto-a-ponto, define-se primeiramente, uma coleção de pontos para o robô. Em seguida, obtém-se a série que é armazenada na memória do controlador. Ao executar o programa, o braço do robô move-se por vários pontos, de acordo com a ordem dos passos definidos na série. Em cada passo o robô sabe seu destino, mas não conhece a trajetória que traçará [7].

Robôs com controle ponto-a-ponto são geralmente usados em séries onde o atuador não precisa realizar nenhuma função no decorrer do movimento. Uma aplicação típica é a solda em ponto [7]. Uma grande maioria dos robôs opera em controle ponto-a-ponto.

Já os robôs de controle de trajetória contínua, podem realizar ciclos de movimento em que a trajetória seguida pelo robô é controlada. Isso é geralmente realizado fazendo com que

o robô se desloque através de uma série de pontos pouco espaçados em relação à trajetória total que descrevem a trajetória desejada. Os pontos individuais são definidos pela unidade de controle e não pelo programador. O movimento linear é uma forma comum de controle por trajetórias contínuas para robôs industriais. O programador especifica o ponto de partida e o ponto final da trajetória, e a unidade de controle calcula a seqüência de pontos individuais que permitem ao robô seguir uma trajetória retilínea. Alguns robôs têm a capacidade de seguir um caminho suave, curvo, que foi definido por um programador que movimenta manualmente o braço através do ciclo de movimento desejado. Atingir um controle por trajetórias contínuas precisas requer que a unidade controladora seja capaz de armazenar um grande número de localizações de pontos individuais que definem a trajetória curva composta. Isso geralmente envolve o uso de um computador digital como controlador de robô. O controle por trajetórias contínuas é requerido para certos tipos de aplicações industriais, tais como pintura e soldagem a arco.

2.11 – MODIFICAÇÕES DE HARDWARE

Para novas implementações, o controle do braço mecânico foi substituído por um circuito integrado programável, o qual veio facilitar a comunicação entre o robô e um computador qualquer, através da porta paralela. O sistema como um todo pode ser dividido em três partes:

- O circuito de potência também conhecido como *driver*;
- O circuito de controle que é composto pelo FPGA[8] (*Field Programmable Gate Arrays*);
- O software desenvolvido em JAVA que comanda o circuito de controle.

[8] Circuitos Reprogramáveis.

Nos próximos itens, será explanado um pouco mais sobre o circuito de potência, o circuito de controle e o software de controle desenvolvido.

2.11.1 – CIRCUITO DE POTÊNCIA

O circuito de potência é responsável por gerar ganho de corrente necessário para acionar todos os motores acoplados ao braço mecânico. Este circuito possui as seguintes especificações:

- Pode ser alimentado com tensões de até 46 V;
- Pode fornecer uma corrente total de 4A, sendo 1A por canal;
- Proteção contra superaquecimento;
- Todo o circuito pode acionar até oito motores, sendo que cada um no máximo de 1A de consumo;
- Seu controle é feito por nível lógico (zero ou um);
- Foi montado com dissipadores de calor;
- A sua fonte de alimentação necessita de pelo menos 3A devido às correntes de recirculação apesar de o circuito ser protegido contra este tipo de corrente.

O circuito é composto por quatro microcontroladores L298, sendo que cada um possui duas pontes completas de transistores bipolares. O diagrama de blocos e a pinagem deste circuito integrado são mostrados nas figuras 2.8 e 2.9 respectivamente.

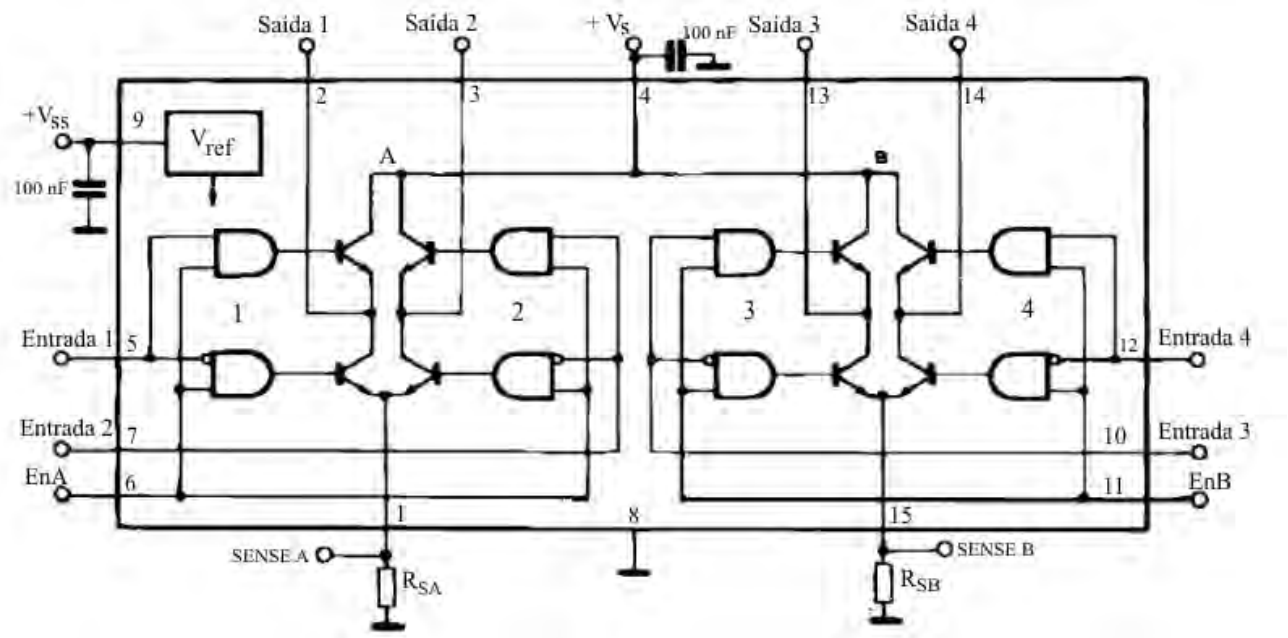


Figura 2.8: Diagrama de blocos do CI L298.

Os pinos do CI da figura 2.9 denominados “INPUT” são provenientes do circuito ALTERA o qual é responsável pelo controle. Os pinos “ENABLE” e “LOGIC SUPPLY VOLTAGE” são ligados em 5 volts e os pinos “CURRENT SENSING” são aterrados.

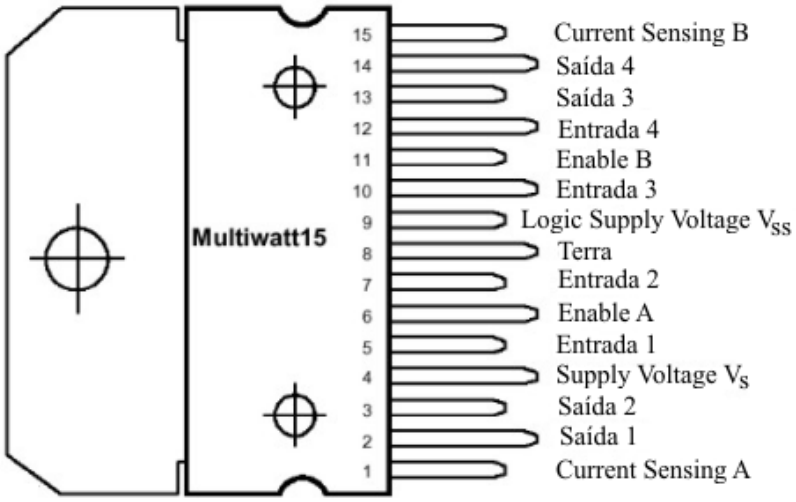


Figura 2.9: Pinagem do CI L298.

O circuito foi montado e testado em laboratório e é mostrado nas figuras 2.10 e 2.11.

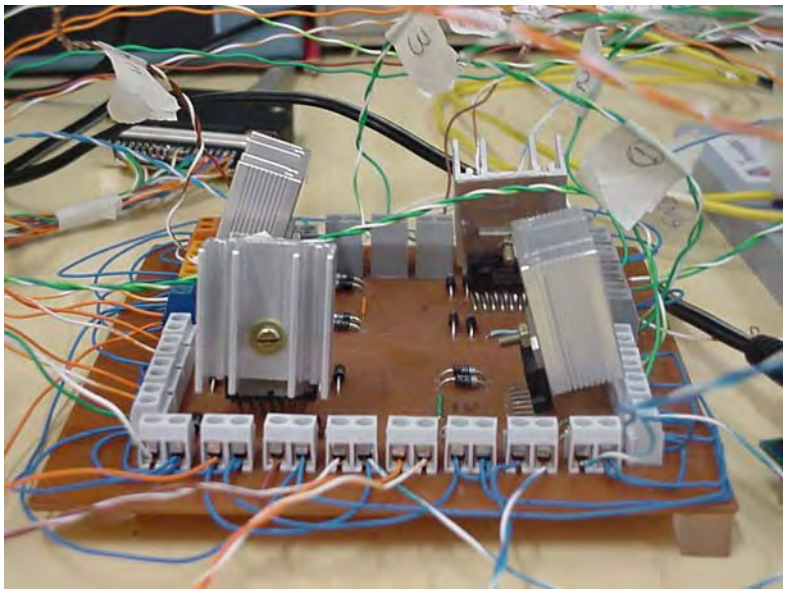


Figura 2.10: Circuito de potência utilizado (a)

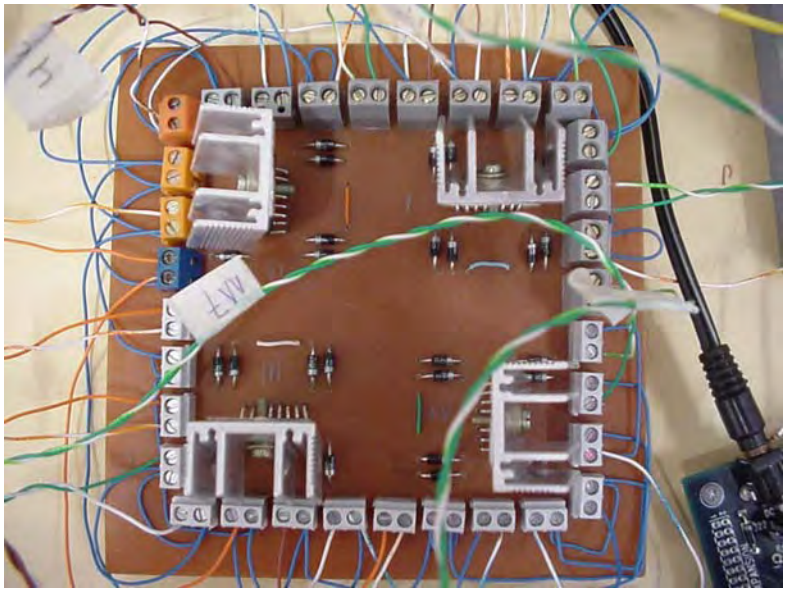


Figura 2.11: Circuito de potência utilizado (b).

2.11.2 – CIRCUITO DE CONTROLE

O circuito integrado foi simulado e implementado no componente MAX EPM7128SLC84-7 ou no componente FLEX EPF10K20RC240-7, disponível no pacote de laboratório do Projeto Universitário da Altera. A linguagem VHDL[9] (*Very High Speed Integrated Circuit Hardware Description Language*) foi utilizada para descrever o comportamento do circuito integrado.

[9] Linguagem de Descrição de Hardware

O circuito de controle gera os sinais lógicos necessários para o circuito de potência acionar corretamente os motores, além de multiplexar os sinais dos *encoders*, enviando-os para a porta paralela do computador.

Os pinos D0 a D4 provenientes da porta paralela do computador são responsáveis pela ativação dos motores por meio da seleção dos bits de cada saída através do circuito de controle. Uma vez obtido esses sinais, o circuito de controle determina qual dos motores terá o *encoder* óptico acionado.

O CI foi programado utilizando a linguagem VHDL, a qual tornou esta etapa muito simples. A programação está inserida no apêndice A. A figura 2.12 ilustra o Kit de programação da ALTERA.

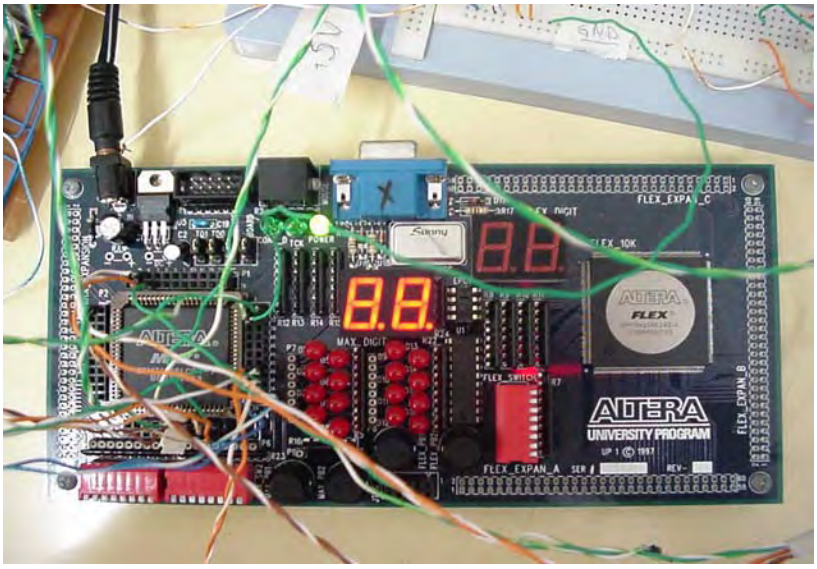


Figura 2.12: Kit da Altera utilizado nos testes.

Na montagem final foi projetada uma placa de circuito impresso onde se utilizou o CI EPM7032LC44-12 também da ALTERA. Este circuito integrado possui 600 portas e até 36 pinos de I/O. Este circuito deve ser alimentado com 5 volts.

O circuito de controle gera os sinais lógicos para o circuito de potência para que os motores sejam acionados na ordem correta além de realizar a multiplexagem dos sinais dos *encoders* para que possam ser enviados a porta paralela do computador.

O roteamento dos pinos do circuito integrado é feito pelo Software Max+Plus II da ALTERA, como mostra a figura 2.13.

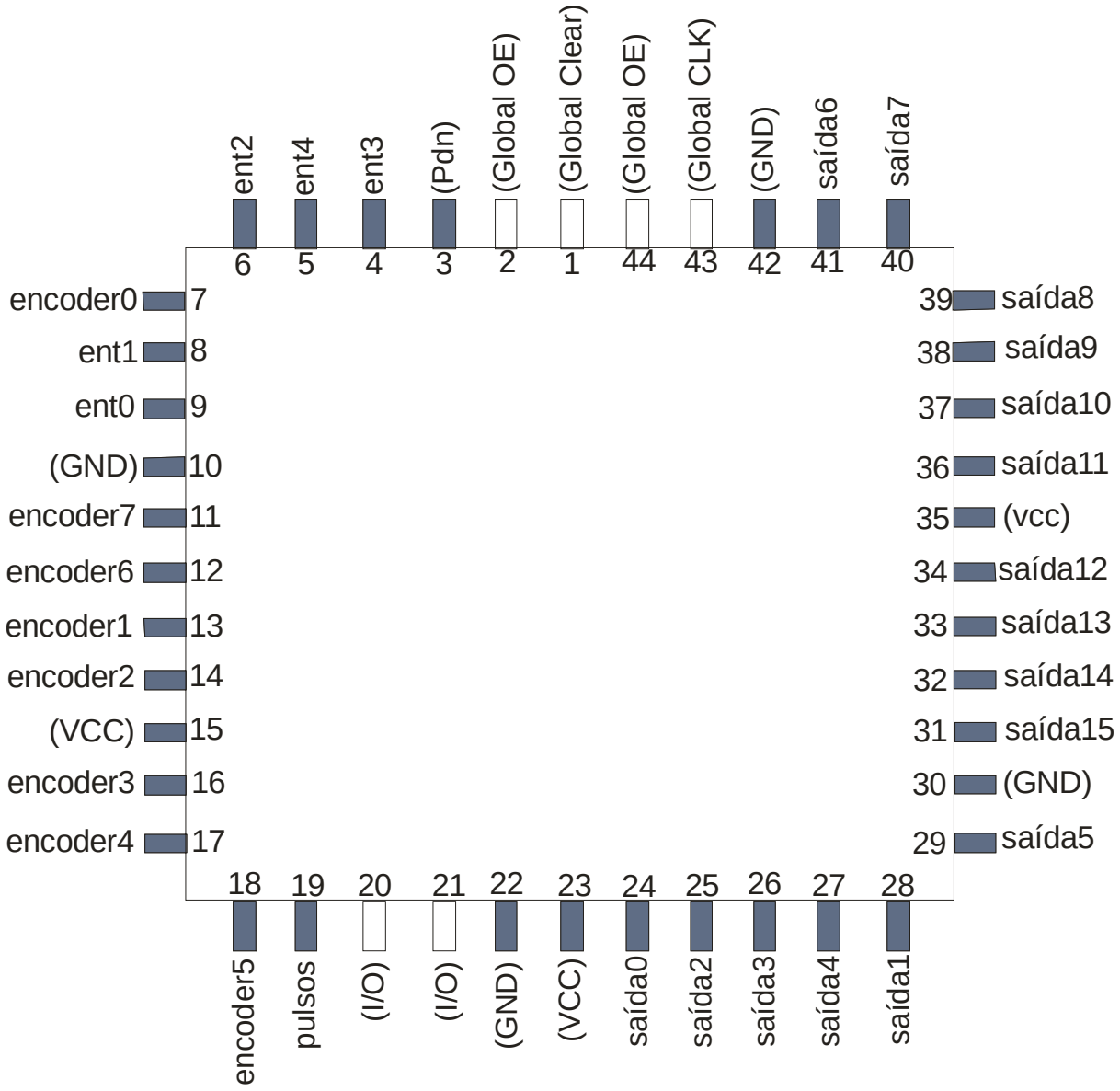


Figura 2.13: Pinagem final do CI EMP7032 fornecida pelo Max+Plus II Profissional.

Os pinos “E0 a E4” são provenientes da porta paralela do computador (D0 a D4). O pino “PULSOS” é a saída já multiplexada dos sinais dos *encoders*, que vai para o pino “INTERRUPT” da porta paralela. Os pinos “ENC0 a ENC7” são as entradas dos sinais dos *encoders* e os pinos “S0 a S15” são as saídas para o circuito de controle.

2.11.3 – SOFTWARE DE COMANDO

O Software de comando foi implementado utilizando a linguagem de programação JAVA, versão 1.4.2 da *Sun Microsystems*. É responsável pelo envio de bits de uma máquina cliente, utilizando a rede de computadores (WWW), para um servidor que está conectado ao circuito de controle determinando o funcionamento de um dos motores pertencentes ao robô, além de receber *streams* de mídia do servidor, tendo como referência seu endereço de IP.

O programa computacional elaborado permite que os motores sejam acionados individualmente, fazendo com que o braço execute o deslocamento no espaço. O acionamento de um motor via máquina cliente, faz com que este seja executado no servidor, através de trocas de mensagens na rede de computadores. Detalhes da comunicação e métodos utilizados para este fim serão mostrados no capítulo seguinte.

CAPÍTULO 3 – HISTÓRICO E CARACTERÍSTICAS DA LINGUAGEM JAVA

Apresenta-se um breve relato histórico sobre a linguagem de programação JAVA. Posteriormente, definem-se as principais características, que motivaram a escolha desta linguagem para a elaboração do software de controle do braço mecânico.

3.1 – HISTÓRICO

Cornell [10], afirma que a linguagem de programação JAVA surgiu da necessidade que a *Sun Microsystems* tinha, de criar um protótipo de um controle remoto que seria utilizado em aparelhos eletrônicos (eletrodomésticos). Surgiu então, o projeto "*Green*", que tinha por objetivo construir este protótipo.

Os *chips* utilizados nestes controles não possuíam muita potência nem memória, e os fabricantes podiam escolher diferentes unidades centrais de processamento. Por isso, era preciso uma linguagem que gerasse um código compacto o qual pudesse ser executado em várias plataformas. Em 1991, James Gosling, programador da *Sun*, criou JAVA [10].

Em 1992, foi lançado o primeiro aparelho, porém, nenhum fabricante estava interessado em produzi-lo. Tentou-se utilizar o produto em um projeto de TV a cabo, e também não se conseguiu acertar nenhum contrato.

Enquanto isso, a WWW crescia cada vez mais. Na época, apenas um *browser*[10] estava sendo utilizado, o Mosaic. Foi aí que a *Sun* decidiu abandonar o *hardware* e adaptar o *software* a trabalhar com a Internet. Foi desenvolvido um *browser* utilizando as características do projeto "*Green*" como: arquitetura neutra, tempo real, confiável e seguro. Dessa forma nasceu o Hotjava, desenvolvido totalmente em JAVA para demonstrar o poder da linguagem.

[10] Programa usado para navegar por páginas da Internet.

Esse *browser* também conseguia interpretar programas escritos em JAVA inseridos em uma página de hipertexto, as *applet*s[11].

Em 1995, a Netscape lançou a versão 2.0 do seu *browser* (Netscape Navigator 2.0) compatível com JAVA, o que favoreceu a divulgação da linguagem.

Apesar de seu apelo ser mais forte na Internet, JAVA pode ser utilizada para construir aplicativos que rodam em arquiteturas diferentes desta, como cliente-servidor ou apenas *stand-alone*[12] [10].

3.2 – CARACTERÍSTICAS

JAVA é uma nova linguagem de programação com elementos de C, C++ e outras linguagens com bibliotecas altamente voltadas para o ambiente da Internet. Não é uma linguagem de programação pequena, e aprendê-la inteiramente requer certo esforço, porém para quem tem algum contato com C ou C++, ou programação orientada a objetos, se sentirá mais à vontade.

Segundo a *Sun Microsystems*, JAVA foi estruturado para ser: "Orientado a objetos, robusto, seguro, independente de arquitetura, alto desempenho, interpretável, com linhas de execução *(thread*s[13]), dinâmico e portável, como já mencionado anteriormente" [10] e [11].

Os autores da linguagem JAVA escreveram um "Informe Oficial" *(White Paper*) que explica os objetivos e realizações do projeto. Este informe está organizado com base em onze termos, os quais são adjetivos que caracterizam a linguagem, atribuídos segundo os motivos e problemas que se propõe a resolver, os quais são relacionados a seguir.

---

[11] Pequeno programa escrito na linguagem JAVA que vem embutido em páginas da web.

[12] São aplicativos completamente auto-suficientes, não necessitando de um segundo software para o seu funcionamento.

[13] Fluxo de controle seqüencial isolado dentro de um programa. Permite que um programa simples possa executar várias tarefas diferentes ao mesmo tempo, independentemente umas das outras.

3.2.1 – SIMPLES

JAVA retirou de sua sintaxe algumas características presentes no C++, como arquivos de cabeçalho, aritmética de ponteiros, estruturas, uniões, entre outras. Algumas destas características são confusas e podem ocasionar problemas na execução de um aplicativo, se o programador não tiver pleno conhecimento do que está fazendo, como, por exemplo, acessar uma porção de memória não destinada a ele.

3.2.2 – ORIENTADO A OBJETO

A principal diferença entre JAVA e C++ no que diz respeito à orientação a objeto, é a herança múltipla, para qual JAVA apresenta uma melhor solução. JAVA suporta herança, mas não herança múltipla. A ausência de herança múltipla pode ser compensada pelo uso de herança e interfaces, onde uma classe herda o comportamento de sua superclasse além de oferecer uma implementação para uma ou mais interfaces.

3.2.3 – DISTRIBUÍDO

A linguagem possui bibliotecas completas de rotinas prontas para implementações envolvendo protocolos baseados em TCP/IP[14] (*Transmission Control Protocol/Internet Protocol*), como HTTP[15] (*HyperText Transfer Protocol*) e FTP[16] (*File Transfer Protocol*), permitindo abrir e acessar objetos na Internet através de URL's[17] (*Uniform Resource Location*) como se estes estivessem na máquina local. Estas bibliotecas facilitam a conexão de computadores através da Internet ou de Intranets coorporativas, diferentemente da programação de rede em C ou C++.

---

[14] Conjunto de padrões da Internet que orienta o tráfego de informações e define o endereçamento e o envio de dados. Para que dois computadores se comuniquem na Internet, é preciso que ambos utilizem o TCP/IP.
[15] Protocolo de comunicação que viabiliza as ligações entre os clientes de WWW e os Web sites.
[16] Protocolo para transferência de arquivos. O FTP pode ser utilizado para copiar arquivos da rede para o computador do usuário e vice versa.
[17] Padrão de endereçamento da Web. Permite que cada arquivo na Internet tenha um endereço próprio, que consiste de seu nome, diretório, máquina onde está armazenado e protocolo pelo qual deve ser transmitido.

3.2.4 – ROBUSTO

JAVA possibilita a escrita de programas que precisam ser confiáveis de várias formas. O compilador faz uma varredura antecipada de possíveis problemas, realiza uma verificação dinâmica (tempo de execução) e elimina situações sujeitas a erros, além de possibilitar a criação de um software robusto, o que significa que o mesmo não "quebra" facilmente devido a erros de programação.

Uma linguagem de programação que encoraja software robusto, geralmente, impõe mais restrições ao programador quando este está escrevendo o código fonte. Estas restrições incluem aquelas com relação aos tipos de dados e ao uso de ponteiros, eliminando, a possibilidade de sobrescrita de memória e conseqüente destruição de dados.

A presença de coleta automática de lixo evita erros comuns que os programadores cometem quando são obrigados a gerenciar diretamente a memória (C, C++, Pascal). A eliminação do uso de ponteiros, em favor do uso de vetores, objetos e outras estruturas substitutivas trazem benefícios em termos de segurança. O programador é proibido de acessar a memória que não pertence ao seu programa, além de não ter chances de cometer erros comuns, tais como uso indevido de aritmética de ponteiros.

Os mecanismos de tratamento de exceções também tornam as aplicações mais robustas, não permitindo que elas abortem, mesmo quando rodando sob condições anormais.

3.2.5 – SEGURO

Como o JAVA foi criado para ambientes de rede/distribuídos, os recursos de segurança receberam muita atenção. Por exemplo, se for executado um binário transferido por *download* da rede, o mesmo poderá estar infectado por vírus. Os aplicativos do JAVA apresentam garantia de resistência contra vírus e de que não serão infectados, pois são incapazes de acessar listas (*heap*s), pilhas (*stacks)* ou a memória do sistema.

Alguns exemplos de segurança do JAVA que supostamente diz o que um programa não pode fazer são:

- Evitar erros de *stack overflow*[18].
- Corromper a memória fora de seu próprio processo.
- Ler ou escrever arquivos locais quando é chamado em um carregador de classes seguro, como um *browser* Web.

A segurança de JAVA será comprometida, se for utilizado algum método nativo na implementação de um programa JAVA. Métodos nativos são rotinas escritas em outras linguagens, que são chamadas a partir do JAVA. Estas rotinas podem não ser tão seguras quanto códigos em JAVA, especialmente se for escrita em uma linguagem como C ou C++ que não oferecem proteção contra o perigo de se sobrescrever memória mediante uso de ponteiros inválidos.

3.2.6 - ARQUITETURA NEUTRA

"O compilador gera um formato de arquivo-objeto com arquitetura neutra – o código compilado pode ser executado em muitos processadores, desde que o sistema local tenha o módulo em tempo de execução de JAVA. O compilador de JAVA faz isso gerando instruções em *bytecodes*[19] que nada têm a ver com uma arquitetura de computador em particular. Em vez disso, eles são projetados para serem fáceis de interpretar em qualquer máquina e facilmente traduzidos em código de máquina nativo no momento da execução" [10].

Derivado da natureza distribuída do cliente/servidor, um importante recurso de projeto da linguagem JAVA é o suporte para clientes e servidores em configurações heterogêneas de

[18] Estouro de Pilha.
[19] O código JAVA é compilado e, este pseudocódigo gerado é formado por *bytecodes*. Os *bytecodes* são, em geral, executados por um outro programa, o interpretador JAVA (JVM – Máquina Virtual JAVA). O interpretador JAVA entende o conjunto lógico de instruções representado pelos *bytecodes*, simulando a arquitetura virtual para a qual foram desenvolvidos.

rede. Devido à portabilidade da linguagem, os objetos binários de código de bytes podem ser executados em plataformas diferentes.

Os objetos de códigos de bytes de arquitetura neutra contêm instruções de computador estéreis; não possuem vínculos ou dependências de nenhuma arquitetura de computador. Em vez disso, as instruções são fáceis de decifrar, seja qual for arquitetura de plataforma, e podem ser convertidas dinamicamente para o código de máquina nativo de qualquer plataforma com relativa facilidade.

Um programa fonte escrito em linguagem JAVA é traduzido pelo compilador para os *bytecodes*, que são interpretados pela Máquina Virtual Java (JVM – *JAVA Virtual Machine*). A vantagem desta técnica é evidente: garantir uma maior portabilidade para os programas JAVA em código-fonte e compilados.

O formato de independência de arquitetura concede grandes benefícios, tanto ao usuário quanto ao programador, pois, uma vez que o código é disponibilizado, pode ser executado em qualquer plataforma.

3.2.7 – INTERPRETADO

O interpretador de JAVA pode executar os *bytecodes* diretamente em qualquer máquina para o qual tenha sido transportado. Esta é uma vantagem enquanto se desenvolve uma aplicação, mas ela é claramente mais lenta que um código compilado. A vantagem é que ela permite um acompanhamento mais apurado no processo de desenvolvimento, na depuração do código.

3.2.8 – ALTO DESEMPENHO

O termo não é correto para descrever a desempenho de um aplicativo ou *applet* JAVA quando leva-se em consideração somente o interpretador. Certamente a velocidade dos *bytecodes* interpretados pode ser aceitável, mas não é rápida.

Compiladores específicos para JAVA não estão disponíveis. Ao invés disso, há compiladores *Just-in-time* (JIT)[20]. Estes trabalham compilando uma vez os *bytecodes* em código nativo. Em seguida lêem o resultado e repetem a compilação caso necessário. Ainda que mais lento que compiladores nativos, os compiladores *Just-in-time* podem aumentar de 10 a 20 vezes a velocidade de muitos programas e são quase sempre significativamente mais rápidos que o interpretador JAVA.

3.2.9 – MULTITHREADED[21]

As linhas de execução em JAVA também conseguem tirar proveito de sistemas multiprocessados, se o sistema operacional básico também o conseguir. Por outro lado, as implementações de *thread* nas principais plataformas diferem bastante, e JAVA não se esforça para ser independente de plataforma com relação a isso.

3.2.10 – DINÂMICO

JAVA foi projetada para adaptar-se a um ambiente em evolução. As bibliotecas podem adicionar livremente novos métodos e variáveis de instância sem qualquer efeito sobre seus clientes. Nesta linguagem, a descoberta do tipo de dado em tempo de execução é imediata.

Esta é uma característica importante nas situações em que o código precisa ser adicionado a um programa em execução. Um exemplo é o código que é transferido da Internet para ser executado no *browse*r.

[20] Compilação imediatamente antes da execução do programa.
[21] *Mutithread* é a capacidade de um programa fazer mais de uma coisa ao mesmo tempo, como por exemplo, gravar um arquivo e imprimir outro.

3.3 – CONCEITOS DE ORIENTAÇÃO A OBJETOS

Linguagens de programação, como os próprios idiomas humanos, evoluem ao longo do tempo. Há um constante refinamento e aperfeiçoamento para atender às exigências cada vez maiores dos usuários. Como outras linguagens de programação modernas, como C++, JAVA é uma mistura de várias técnicas desenvolvidas ao longo dos anos [12].

Assim, seguindo os paradigmas de desenvolvimento mais utilizados, a linguagem JAVA é orientada a objetos. O paradigma de orientação a objetos já demonstrou que é muito mais do que uma simples “moda”. Benefícios da tecnologia de objetos são observados à medida que um número cada vez maior de empresas “migram” seus produtos para este tipo de modelo.

Infelizmente, o conceito de “orientação a objetos” continua um pouco confuso e é propagado como o que vai resolver todos os problemas do mundo do software. Por exemplo, uma visão superficial de programação orientada a objetos é afirmar que este paradigma é simplesmente uma nova maneira de organizar o código fonte. Na verdade, pode-se alcançar com técnicas de programação orientada a objetos, resultados que não seriam possíveis de se obter com técnicas procedurais.

Como uma linguagem orientada a objetos, JAVA aproveita os melhores conceitos e funcionalidades de linguagens mais antigas (principalmente Eiffel, SmallTalk, Objective C e C++). JAVA vai além do C++ pois estende o modelo de objetos e remove as maiores complexidades da linguagem. Por exemplo, com exceção dos tipos de dados primitivos, tudo em JAVA é um objeto (até mesmo os tipos primitivos podem ser encapsulados dentro de objetos, se for necessário).

3.3.1 CONCEITOS BÁSICOS

Nos próximos itens deste capítulo será definido o conceito de objetos, mensagens e classes, posteriormente algumas definições sobre encapsulação, herança e polimorfismo. Todos estes conceitos são partes essenciais da Linguagem de Programação proposta, pois é uma linguagem totalmente orientada a objetos.

3.3.1.1 – OBJETOS

Como o próprio nome "orientado a objetos" indica, o conceito de objetos é fundamental para o entendimento desta tecnologia. Pode-se olhar a sua volta e encontrar vários exemplos de objetos: cachorro, cadeira, televisão, bicicleta, entre outros.

Estes objetos do mundo real compartilham duas características: possuem estados (propriedades) e tem um comportamento. Um cachorro tem uma propriedade (nome, cor, raça, com fome) e um comportamento (latindo, comendo, lambendo). Analogamente, objetos de Software são modelados de acordo com os objetos do mundo real, ou seja, possuem também estados e comportamento. Um objeto de software armazena seu estado em variáveis e implementa seu comportamento com métodos.

Portanto, podem-se representar objetos do mundo real utilizando objetos de software. Além disto, objetos de software podem ser usados para modelar conceitos abstratos, como por exemplo, um evento de interface gráfica que representa a ação do usuário pressionando uma tecla.

A ilustração da figura 3.1 é uma representação comum do conceito de objeto:

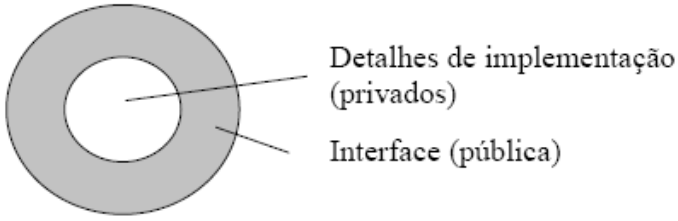


Figura 3.1: Representação de objeto.

Tudo que o objeto de software conhece (estado) e pode fazer (comportamento) é expresso pelas variáveis e métodos do objeto. Um objeto que modela uma bicicleta poderia ter variáveis que indicam seu o estado atual: velocidade é 20 km/h, marcha atual é a quinta. Exemplos de métodos deste objeto seriam: frear, aumentar a velocidade da pedalagem e trocar de marcha. Estas variáveis e métodos são formalmente chamados de variáveis de instância e métodos de instância a fim de distingui-los de variáveis e métodos de classe descritos no item Classes.

As variáveis de um objeto fazem parte do seu núcleo (centro). Os métodos envolvem e escondem o núcleo do objeto de outros componentes (objetos) da aplicação. Empacotar as variáveis de um objeto sobre a proteção de seus métodos é chamado de encapsulação (seção). Basicamente, a encapsulação é usada para esconder detalhes de implementação pouco importantes. Por exemplo, quando se deseja trocar uma marcha na sua bicicleta, não é preciso saber como o mecanismo de troca de marchas funciona apenas qual alavanca deve ser movida. Analogamente, em programas, muitas vezes não é necessário saber como um objeto foi implementado, apenas saber qual método deve ser invocado. Assim, os detalhes de implementação podem ser mudados sem afetar outras partes do programa.

Esta representação conceitual de um objeto, um núcleo de variáveis dentro de uma membrana protetora de métodos, é um modelo ideal que deve servir de meta para projetistas de sistemas orientados a objetos. Entretanto, esta representação não é totalmente realista.

Muitas vezes, por razões de implementação ou eficiência, um objeto pode desejar expor algumas de suas variáveis ou esconder alguns de seus métodos.

3.3.1.2 – MENSAGENS

Geralmente, um objeto sozinho não é muito útil e aparece como um componente de uma aplicação maior que contém vários outros objetos. A interação entre os objetos é que permite se obter todas as funcionalidades de uma aplicação. Por exemplo, uma bicicleta só é útil quando outro objeto (ciclista) interage com ela.

Objetos comunicam-se entre si por meio do envio de mensagens. Quando um objeto A deseja que o objeto B realize um dos seus métodos (de B), o objeto A envia uma mensagem para o objeto B.

Algumas vezes, o objeto que recebe a mensagem precisa de mais informações para saber exatamente o que fazer. Por exemplo, quando você quer trocar as marchas em uma bicicleta, devesse indicar qual a marcha desejada. Esta informação acompanha a mensagem como um parâmetro.

Assim, três componentes fazem parte de uma mensagem:

- ✓ O objeto para onde a mensagem é endereçada (bicicleta);
- ✓ O nome do método a realizar (mudar a marcha);
- ✓ Parâmetro(s) necessário(s) para realizar o método (segunda marcha).

Objetos não precisam estar no mesmo processo ou na mesma máquina para enviar e receber mensagens de uns para os outros.

3.3.1.3 – CLASSES

No mundo real, muitas vezes existem vários objetos do mesmo tipo. Por exemplo, utilizando a terminologia de orientação a objetos, pode-se dizer que um objeto bicicleta é uma

instância de uma classe de objetos conhecida como bicicletas. Bicicletas possuem estados e comportamentos comuns. Entretanto, o estado de cada bicicleta é independente e pode ser diferente de outras bicicletas.

Em software orientado a objetos, é possível ter vários objetos do mesmo tipo que compartilham características. Desta forma, pode-se tirar vantagem de que os objetos do mesmo tipo são similares e criar uma "fôrma" para estes objetos. Tais fôrmas de software são chamadas de classes.

Assim, uma classe é uma fôrma (protótipo) que define as variáveis e métodos comuns a todos os objetos de certo tipo.

Valores de variáveis de instância existem para cada instância (objeto) da classe. Assim, depois de criar a classe bicicleta, deve-se instanciá-la a fim de utilizar seus objetos.

Quando se cria uma instância de uma classe, cria-se um objeto daquele tipo e o sistema aloca memória para as variáveis de instância definidas para a classe. Depois de criado, podem-se invocar os métodos de instância do objeto.

Além das variáveis e métodos de instâncias, classes podem também definir variáveis de classe (*class variables)* e métodos de classe (*class methods*). Pode-se acessar variáveis e métodos de classe sem ter a necessidade de se instanciar um objeto da classe. Métodos de classe só podem manipular variáveis de classe, ou seja, não podem acessar métodos ou variáveis de instância.

O sistema cria uma única cópia de uma variável de classe, ou seja, todos os objetos daquela classe compartilham as mesmas variáveis de classe.

3.3.2 – CARACTERÍSTICAS DA TECNOLOGIA DE OBJETOS

Para ser considerada verdadeiramente orientada a objetos, uma linguagem de programação deve oferecer, no mínimo, estas três características:

- ✓ Encapsulação;
- ✓ Herança;
- ✓ Polimorfismo.

3.3.2.1 – ENCAPSULAÇÃO

Uma das principais diferenças entre o paradigma de programação estruturada e o modelo de orientação a objetos é a característica de encapsulação. Encapsulação permite esconder, dentro de um objeto, tanto suas variáveis quanto os métodos que manipulam estas variáveis.

Assim, é possível controlar acessos de outros componentes aos dados do objeto.

Na programação estruturada também pode-se esconder os dados dentro de uma função simplesmente criando variáveis locais. O problema surge quando se deseja tornar uma variável disponível a outras funções. Em um programa estruturado, a solução é criar variáveis globais. Infelizmente, desta forma qualquer função dentro do programa pode acessar os dados.

Na programação orientada a objetos, consegue-se criar variáveis de instância, que podem ser acessadas por qualquer método do objeto, mas não são visíveis por outros objetos.

A idéia de encapsulação, apesar de simples, é uma ferramenta poderosa que oferece dois benefícios principais aos programadores:

- ✓ Modularidade: o código fonte de um objeto pode ser escrito e mantido independentemente do código fonte de outros objetos.
- ✓ Ocultamento da informação: um objeto possui uma interface pública que outros objetos podem utilizar para comunicar-se com ele. Além disto, o objeto pode manter informações e métodos privados que podem ser mudados a qualquer momento sem afetar os outros objetos.

3.3.2.2 – HERANÇA

De maneira geral, objetos são definidos em termos de classes. Consegue-se obter muitas informações sobre um objeto conhecendo a sua classe. Por exemplo, você pode não saber o que é uma *penny-farthing*, mas se lhe disserem que é uma bicicleta, você consegue imaginar o objeto (duas rodas, pedais, assento, etc.).

Sistemas orientados a objetos permitem também que classes sejam definidas em termos de outras classes. Por exemplo, *mountain bikes* e bicicletas de corrida são diferentes tipos de bicicletas. Na terminologia de orientação a objetos, *mountain bikes* e bicicletas de corrida são subclasses da classe bicicleta. Analogamente, a classe bicicleta é uma superclasse de *mountain bikes* e bicicletas de corrida, como mostra a figura 3.2.

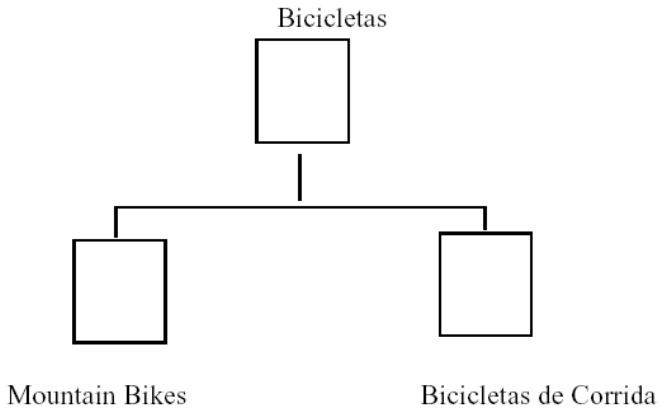


Figura 3.2: SuperClasses e Subclasses

Cada classe herda o estado (na forma das declarações de variáveis) da superclasse. *Mountain bikes* e bicicletas de corrida compartilham alguns estados (velocidade, por exemplo). Além disto, cada subclasse herda os métodos da superclasse. *Mountain bikes* e bicicletas de corrida compartilham certos comportamentos (frear, por exemplo). Entretanto, subclasses não estão limitadas ao comportamento herdado de sua superclasse.

Subclasses podem adicionar variáveis e métodos a aqueles herdados.

Subclasses também podem redefinir (*override*) métodos herdados e oferecer implementações especializadas para estes métodos. Por exemplo, se uma *montain bike* possui um conjunto extra de marchas, você poderia sobrecarregar o método “mudar marcha” de tal forma que seja possível escolher este novo conjunto de marchas.

O conceito de herança pode ser aplicado para mais de um nível. A árvore de herança, ou hierarquia de classe pode ser tão “profunda” quanto necessário. Os métodos e variáveis são herdados através dos níveis. Em geral, quanto mais baixa na hierarquia for a posição de uma classe, mais especializado é o seu comportamento.

Como benefícios do conceito de herança, podemos citar:

- ✓ Subclasses oferecem comportamentos especializados a partir de elementos básicos comuns, oferecidos pela superclasse. Por meio da utilização de herança, programadores podem reusar o código da superclasse várias vezes.
- ✓ Programadores podem definir classes abstratas que determinam comportamentos “genéricos”. A superclasse abstrata define e pode implementar parcialmente o comportamento de uma classe, mas a maioria das informações da classe fica indefinida ou não é implementada. Outros programadores completam estes detalhes por meio de subclasses especializadas.

3.3.2.3 – POLIMORFISMO

Segundo a terminologia de orientação a objetos, polimorfismo significa que uma mesma mensagem enviada a diferentes objetos resulta em um comportamento que é dependente da natureza do objeto que está recebendo a mensagem.

Existem vários tipos de polimorfismo. O primeiro tipo, descrito no item anterior, é quando uma classe redefine a implementação de um método herdado. Este polimorfismo é classificado como polimorfismo de inclusão.

Se a subclasse oferece um método de assinatura (nome, parâmetro de retorno e argumentos) parecida com a do método herdado então não se trata de uma redefinição e sim de uma sobrecarga**,** pois criou-se, na verdade, um novo método.

Outro tipo bastante interessante de polimorfismo é conhecido como acoplamento dinâmico (*dynamic binding*). Por acoplamento, entenda-se a escolha correta de um método a ser executado para uma variável declarada como de uma classe, mas que pode conter um objeto de uma subclasse desta. Por dinâmico, entenda-se tempo de execução.

Devido às características de herança, pode-se atribuir um objeto de uma subclasse a uma variável da classe pai, mas não o contrário. Com o acoplamento dinâmico, é possível fazer com que a execução de um método dependa do tipo do objeto atribuído a variável.

Estas características proporcionam maior flexibilidade para a implementação do Software de Controle, pois JAVA independe da plataforma de trabalho, somando-se ainda o fato de JAVA ser uma linguagem gratuita (*freeware*).

No capítulo seguinte, são mostrados os conceitos empregados para a construção do Software de Controle do braço mecânico utilizando a linguagem de programação JAVA.

CAPÍTULO 4 - IMPLEMENTAÇÃO DO SOFTWARE DE CONTROLE

Para o desenvolvimento do software de controle remoto do braço mecânico SCORBOT ER - III, utilizou-se à linguagem de programação JAVA versão 1.4.2 da *Sun Microsystems* e alguns métodos e bibliotecas desta linguagem, como o uso de Invocação de Métodos Remotos (RMI – *Remote Methods Invocation*), Métodos Nativos (JNI – *JAVA Native Interface*), Conectividade com Banco de Dados JAVA (JDBC – *JAVA Database Connectivity*) e JMF (*JAVA Media Frameworks*) para o monitoramento em tempo real do robô utilizando o protocolo RTP – *Real Time Protocol* (Protocolo de Tempo Real).

A figura 4.1 ilustra o aplicativo cliente que é responsável pelo envio de tarefas ao servidor. Este solicita quais os motores que serão ativados através dos botões de controle, e assim, o envio de tarefas ao servidor é feito com base nos conceitos de RMI. A aquisição dos *frames* no software cliente ocorre através dos conceitos de JMF/RTP, onde é passado à rede, o endereço IP do *host* que estará recebendo os quadros e em qual porta isso ocorrerá.

A figura 4.2 ilustra o aplicativo servidor que é responsável pelo envio de imagens ao cliente e pela execução das tarefas solicitadas. O servidor é responsável pelo controle do robô, pois o braço mecânico se encontra conectado a ele através da porta paralela do computador. Em posse de uma solicitação do cliente através de RMI, o servidor executa a tarefa solicitada utilizando conceitos de JNI. As imagens adquiridas por uma Webcam instalada no servidor são enviadas a um endereço IP de uma máquina cliente específica. Esta receberá os quadros (imagens) através de uma porta de conexão com a rede de computadores, através de JMF/RTP.

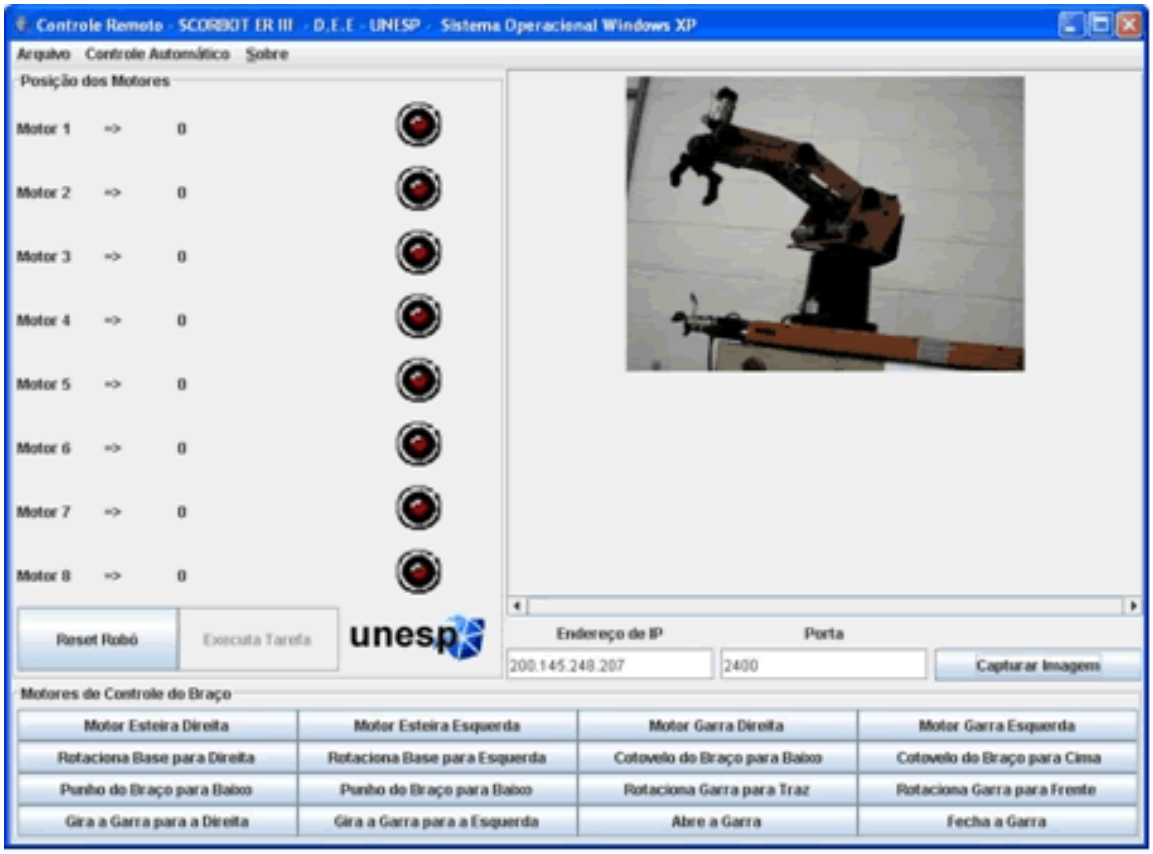


Figura 4.1: Layout do Aplicativo Cliente

Figura 4.2: Layout do Aplicativo Servidor

Na figura 4.3 apresenta-se o *layout* do formulário de cadastro de movimentos do robô. Este formulário possibilita a inserção, remoção, alteração e navegação entre registros do banco de dados Motores.fdb, o qual armazena os valores para que o robô execute uma tarefa pré-programada.

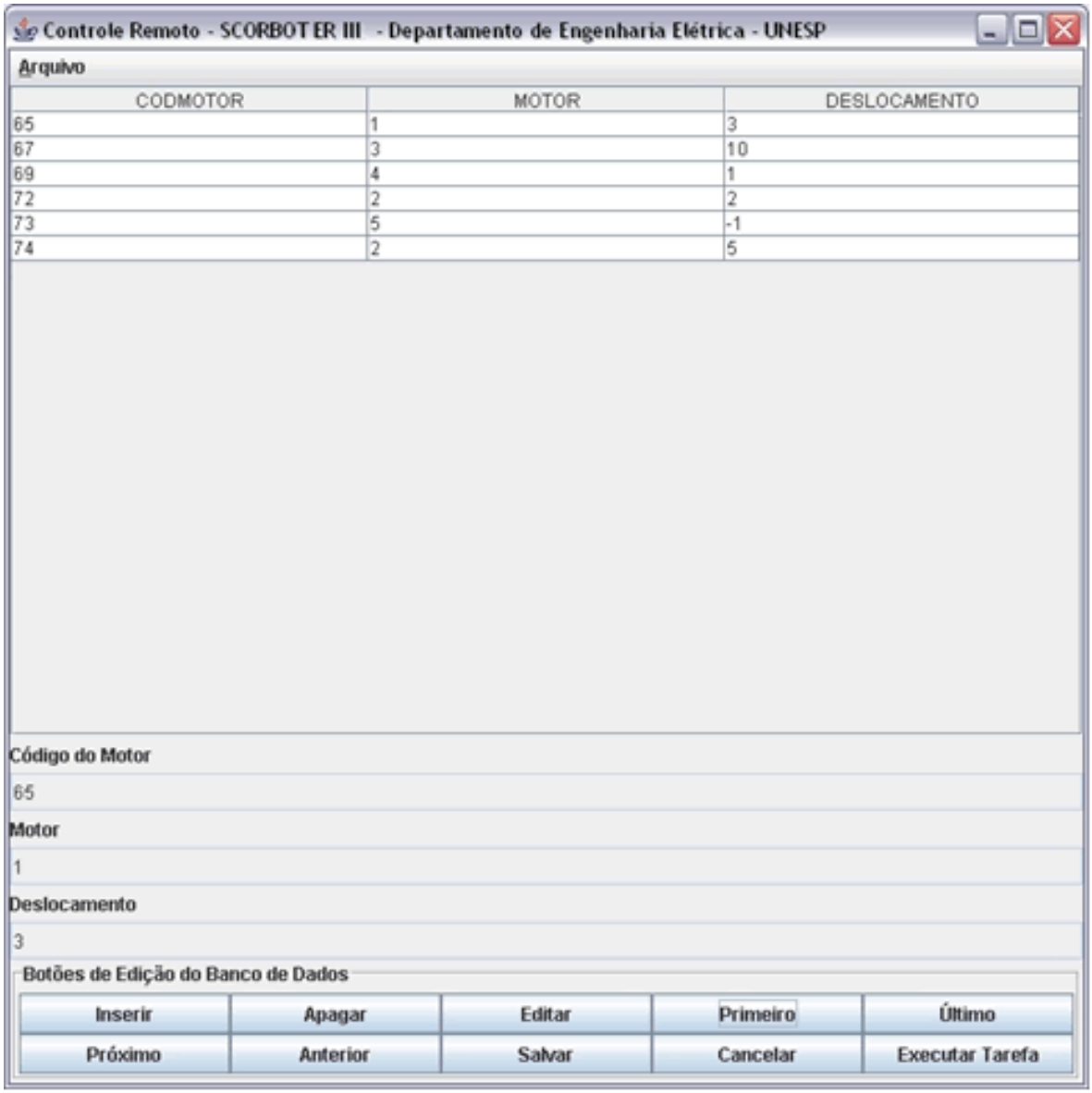


Figura 4.3: *Layout* do formulário de cadastro para o movimento do motor.

Os *layouts* apresentados nas figuras 4.4, 4.5 representam as caixas de diálogo para a inserção de valores no banco de dados.

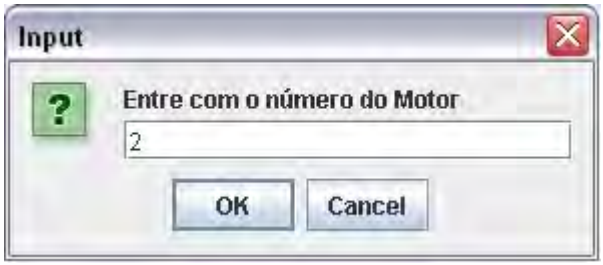


Figura 4.4: Caixa de diálogo para inserir um motor do banco de dados.

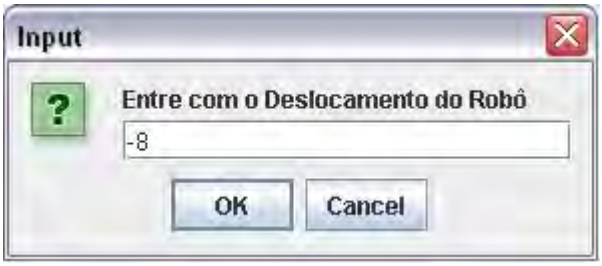


Figura 4.5: Caixa de diálogo para fornecer o deslocamento do robô.

O *layout* da figura 4.6 apresenta a caixa de diálogo para a remoção de uma linha, correspondente às coordenadas de deslocamento de um determinado motor, do banco de dados.

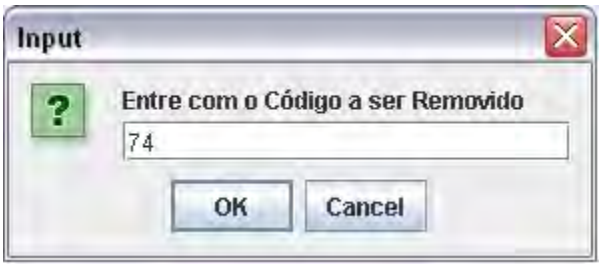


Figura 4.6: Caixa de diálogo para remover às coordenadas de um motor do banco de dados.

Na figura 4.7, ilustra-se a tela de informações com respeito ao desenvolvimento do Sistema de Controle do SCORBOT ER –III.

A seguir serão detalhados os procedimentos para a criação do programa que tem por objetivo à manipulação remota do braço mecânico do robô SCORBOT ER – III, utilizando a rede mundial de computadores através da linguagem de programação JAVA. Uma cópia dos códigos fonte do software de controle do SCORBOT está disponível no final desta dissertação.

Como dito anteriormente, uma das vantagens de se trabalhar com aplicações JAVA é a sua portabilidade. Alguns autores utilizam uma técnica denominada *Common Gateway Interface* (CGI) para interpretar as entradas do usuário, por meio de um navegador, e retornar estas informações baseadas nessas entradas [13]. Esta técnica torna-se mais cômoda para os

desenvolvedores, pois não exige conhecimentos de programação nas camadas de protocolo de rede, Internet e WEB.

Figura 4.7: *Layout* do formulário sobre o sistema.

4.1 – MÉTODOS NATIVOS

O JAVA é uma linguagem relativamente nova, significando que possam existir situações em que será necessário integrar programas feitos em JAVA com serviços já existentes escritos em outras linguagens. A plataforma JAVA proporciona o JNI para auxílio em integrações de diferentes linguagens [14].

A biblioteca JNI nos auxilia na integração destas linguagens fazendo a especificação de como é que se nomeia (dar nome aos métodos JAVA) e se invoca esses serviços de forma a que o *JAVA Virtual Machine* (JVM) possa localizá-los [14].

Existem três razões óbvias pelas quais o uso de Métodos Nativos pode ser a escolha certa:

- Possuir volumes substanciais de código testado e depurado disponíveis em outra linguagem. Portar o código para a linguagem de programação JAVA seria perda de tempo, e o código resultante precisaria ser testado e depurado novamente;
- Seu aplicativo exige acesso a recursos de sistema ou dispositivos e usar tecnologia JAVA seria complicado, na melhor das hipóteses, ou impossível, na pior das hipóteses;
- Maximizar a velocidade do código é essencial. Por exemplo, a tarefa pode ter o tempo como um componente importante, ou então, pode ser que o código seja usado com tanta freqüência, que aperfeiçoá-lo seria custoso. Na verdade, este é o motivo menos plausível. Com a compilação *just-in-time* (JIT), os cálculos pesados codificados na linguagem de programação JAVA não são muito mais lentos do que um código compilado em C.

Para o acesso a porta paralela e controle do robô, foi utilizado um método nativo programado em C++, o qual possui duas funções chamadas "*outportb*" e "*inportb*", encarregada pelo envio de *bits* à porta paralela para ativação dos motores do robô e controle dos *encoders* nos localizados junto aos motores do robô, já que a utilização da biblioteca JAVAX.COMM, específica da Linguagem JAVA, o controle tornou-se insatisfatório, uma vez que, esta biblioteca não é portável a todas as plataformas computacionais existentes [15].

Portanto, é possível invocar código escrito em qualquer linguagem a partir de um programa em JAVA. Para que essa invocação possa ser efetuada são necessários os seguintes passos:

- Declaração de métodos nativos ("JAVA Native Methods");
- Carregamento da biblioteca que contém o código nativo (na plataforma Windows estas assumem a forma de "Dynamic Link *Libraries*" – DLL's);

- Chamada dos métodos nativos.

Como mencionado anteriormente, primeiramente precisa-se declarar o método nativo em uma classe. Neste caso, a classe pertencente ao projeto definida como *ControlaMotores.class* e, arquivo de fonte com nome *ControlaMotores.java* mostrada na tabela 1 define-se o método.

Tabela 1: Implementação da classe *ControlaMotores.java.*

```
class ControlaMotores
{
        public static native void controle(int NumMotor, int PassoMotor);
        static
        {
                System.loadLibrary("Controla");
        }
}
```

A palavra-chave *native* alerta ao compilador que o método será definido externamente. Nestes métodos, não aparecerão códigos de programação JAVA, e o cabeçalho do método será seguido imediatamente por um ponto-e-vírgula de terminação.

Depois de definida, compilou-se a classe *ControlaMotores.class* executando o utilitário *javah*, para a criação automática das assinaturas que deverão ser utilizadas na implementação do código nativo.

Para usar *javah*, primeiro compila-se o arquivo fonte da seguinte maneira:

- *javac ControlaMotores.java*

Em seguida, é chamado o utilitário *javah* para produzir o arquivo de cabeçalho que será anexado ao código em C.

- *javah ControlaMotores*

Depois de compilado com êxito, o uso de *javah* cria um arquivo de cabeçalho *ControlaMotores.h*, como mostra a tabela 2:

Tabela 2: Arquivo de Cabeçalho *ControlaMotores.h*

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class ControlaMotores */

#ifndef _Included_ControlaMotores
#define _Included_ControlaMotores
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     ControlaMotores
 * Method:    controle
 * Signature: (II)V
 */
JNIEXPORT void JNICALL Java_ControlaMotores_controle
  (JNIEnv *, jclass, jint, jint);

#ifdef __cplusplus
}
#endif
#endif
```

O arquivo "*ControlaMotores.h*" define uma interface que faz a correspondência entre os métodos escritos em JAVA com as funções nativas escritas em C++. Neste exemplo, a assinatura do "Controle" faz corresponder os argumentos e resultados do método Controle implementada na biblioteca de código nativo. As assinaturas deste exemplo são mostradas na tabela 3.

Tabela 3: Assinaturas criadas pelo arquivo de cabeçalho.

```
/*
 * Class:     ControlaMotores
 * Method:    controle
 * Signature: (I)V
 */
JNIEXPORT void JNICALL java_ControlaMotores_controle
(JNIEnv *, jclass, jint);
```

Os parâmetros da assinatura do método têm o seguinte significado:

- JNIEnv *: Um apontador para a ambiente interface nativa JAVA.
- jclass: Referência para o objeto que chamou "este" código nativo.

4.1.1 – IMPLEMENTAÇÃO DO MÉTODO NATIVO

No arquivo de código-fonte nativo, deve-se copiar o protótipo da função do arquivo de cabeçalho e fornecer o código da implementação para a função mostrada no método nativo *Controla.c*. Esta função tem a finalidade de acionar através da sub-rotina chamada *outportb*, os motores que compõem o robô e controlar os *encoders* através da função *inportb* do SCORBOT ER - III.

4.1.2 – COMPILAÇÃO DA BIBLIOTECA

A biblioteca necessita ser compilada de forma dinâmica para que possa ser carregada durante o tempo de execução do programa em JAVA ("*runtime*"). Para a compilação da biblioteca utiliza-se o seguinte comando:

- bcc32 -Ic:\j2sdk1.4.2_04\include -Ic:\j2sdk1.4.2_04\include\win32 -LD Controla.c -oControle.dll

O comando "bcc32.exe" pertence ao compilador C++ 5.0 da Borland, e os parâmetros restantes especificam os diretórios necessários "header files". O parâmetro "LD" especifica que o resultado é uma biblioteca dinâmica.

Depois de compilado o arquivo *Controla.c* é gerada automaticamente a biblioteca dinâmica *Controle.dll*.

4.2 – INVOCAÇÃO DE MÉTODOS REMOTOS

Trabalhar com métodos remotos implica em possuir um conjunto de objetos colaborativos, que possam ser localizados em qualquer lugar. Esses objetos comunicam-se através de protocolos padrões de uma rede. Por exemplo, o objeto cliente envia uma mensagem para um objeto no servidor, contendo os parâmetros da solicitação. O objeto servidor reúne as informações solicitadas ou se necessário, comunica-se com objetos adicionais. Uma vez que o objeto servidor tenha a resposta para o cliente, este retorna as informações.

4.2.1 – CHAMADAS DE OBJETOS REMOTOS

O mecanismo RMI permite o acesso a um objeto em uma máquina diferente, sendo chamado de métodos de objeto remoto. Os parâmetros do método devem ser enviados para uma determinada máquina e após a execução dos parâmetros no objeto remoto, envia-se um valor de retorno para o objeto cliente [14].

Como mostra a figura 4.8, o cliente busca informações no servidor, através do método remoto *find*, este método retorna um objeto para o cliente.

Figura 4.8: Chamando um objeto remoto em um objeto servidor

Na terminologia RMI, o objeto cujo método faz a chamada remota é denominado *cliente* e o objeto remoto é chamado de *servidor*. É importante lembrar que a terminologia cliente/servidor aplica-se apenas a uma chamada de método simples. O computador que está executando o código na linguagem de programação JAVA e chama o método remoto, é o cliente dessa chamada e o computador que contém o objeto que processa a chamada, é o servidor. Em qualquer momento, é possível que os papéis sejam invertidos.

No aplicativo de controle do braço mecânico, o cliente passa um parâmetro informando ao servidor qual motor deverá ser executado através de chamadas RMI. O servidor é encontrado uma vez que o mesmo possui uma nomenclatura que o diferencia dentro da rede. Depois de encontrado e de posse do parâmetro enviado pelo cliente, o servidor

executa os comandos para a ativação dos motores fazendo uso de uma biblioteca dinâmica compilada em C++ e acessada através de terminações nativas – JNI.

4.2.2 – ESTABELECENDO CHAMADAS DE MÉTODOS REMOTOS

Executar o exemplo mais simples de objeto remoto exige muito mais configuração do que executar uma *applet* ou um programa independente. Devem-se executar programas nos computadores denominados servidor e cliente. As informações necessárias do objeto devem separar as interfaces do lado do cliente e as implementações no lado do servidor, no qual, através de pesquisas, o cliente localiza objetos no servidor.

Para o funcionamento do código por meio de métodos remotos, ambas as máquinas (servidor/cliente) deverão ter os serviços de rede disponíveis, sendo assim, é necessário o funcionamento do protocolo TCP/IP.

Portanto, neste trabalho, utilizando o conceito de Invocação de Métodos Remotos, a máquina Cliente pode acessar o Servidor, no qual, o robô está conectado e fazer a solicitação de uma tarefa para a sua manipulação. A comunicação entre o Servidor e o robô é feita através da porta paralela do micro-computador.

4.2.3 – INTERFACES E IMPLEMENTAÇÕES (RMI)

O programa cliente necessariamente manipula objetos do servidor. Como esses objetos residem no Servidor, é necessário que esses recursos sejam expressos em uma interface que é compartilhada entre o cliente e o servidor e, portanto, residem simultaneamente nas duas máquinas.

Para a comunicação entre o cliente e o servidor na manipulação do braço mecânico, foi criada um código de interface chamada *InterfaceRMI.java* mostrada na tabela 4:

Tabela 4: Implementação da classe *InterfaceRMI.java*

```
import java.rmi.*;

public interface InterfaceRMI extends Remote {
```

```
void Motor1() throws RemoteException;
void Motor2() throws RemoteException;
void Motor3() throws RemoteException;
void Motor4() throws RemoteException;
void Motor5() throws RemoteException;
void Motor6() throws RemoteException;
void Motor7() throws RemoteException;
void Motor8() throws RemoteException;
void Motor9() throws RemoteException;
void Motor10() throws RemoteException;
void Motor11() throws RemoteException;
void Motor12() throws RemoteException;
void Motor13() throws RemoteException;
void Motor14() throws RemoteException;
void Motor15() throws RemoteException;
void Motor16() throws RemoteException;
void Motor17() throws RemoteException;
void Motor18() throws RemoteException;

void Motor1par(int x) throws RemoteException;
void Motor2par(int x) throws RemoteException;
void Motor3par(int x) throws RemoteException;
void Motor4par(int x) throws RemoteException;
void Motor5par(int x) throws RemoteException;
void Motor6par(int x) throws RemoteException;
void Motor7par(int x) throws RemoteException;
void Motor8par(int x) throws RemoteException;
void Motor9par(int x) throws RemoteException;
void Motor10par(int x) throws RemoteException;
void Motor11par(int x) throws RemoteException;
void Motor12par(int x) throws RemoteException;
void Motor13par(int x) throws RemoteException;
void Motor14par(int x) throws RemoteException;
void Motor15par(int x) throws RemoteException;
void Motor16par(int x) throws RemoteException;
void Motor17par(int x) throws RemoteException;
void Motor18par(int x) throws RemoteException;}
```

Todas as interfaces de objetos remotos devem possuir a interface *Remote* definida no pacote *java.rmi.* Os métodos dessa interface também devem declarar que lançarão uma *RemoteException,* pois as chamadas de métodos remotos são menos confiáveis do que as chamadas locais, podendo ocorrer falha em uma chamada remota. Por exemplo, o servidor ou a conexão de rede pode estar temporariamente indisponível, ou pode haver um problema de rede, assim, o código cliente deve estar preparado para estas possibilidades.

Depois de definido as interfaces, implementou-se no lado do servidor, a classe que efetivamente executará os métodos anunciados na interface remota. Para o programa de controle do robô, foi implementado um código chamado *ServidorRMI.java* como mostrado na tabela 5:

Tabela 5: Implementação da Classe *ServidorRMI.java*

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class ServidorRMI extends UnicastRemoteObject
implements InterfaceRMI
{
    public ServidorRMI() throws RemoteException
    {          }

    public void Motor1() throws RemoteException
    {
            ControlaMotores.controle(1,0);
    }

    public void Motor2() throws RemoteException
    {
            ControlaMotores.controle(2,0);
    }

    public void Motor3() throws RemoteException
    {
            ControlaMotores.controle(3,0);
    }

    public void Motor4() throws RemoteException
    {
            ControlaMotores.controle(4,0);
    }

    public void Motor5() throws RemoteException
    {
            ControlaMotores.controle(5,0);
    }

    public void Motor6() throws RemoteException
    {
            ControlaMotores.controle(6,0);
    }

    public void Motor7() throws RemoteException
    {
            ControlaMotores.controle(7,0);
    }

    public void Motor8() throws RemoteException
    {
            ControlaMotores.controle(8,0);
    }

    public void Motor9() throws RemoteException
    {
            ControlaMotores.controle(9,0);
    }

    public void Motor10() throws RemoteException
    {
            ControlaMotores.controle(10,0);
    }

    public void Motor11() throws RemoteException
    {
            ControlaMotores.controle(11,0);
    }

    public void Motor12() throws RemoteException
    {
            ControlaMotores.controle(12,0);
    }
```

```
        public void Motor13() throws RemoteException
        {
                ControlaMotores.controle(13,0);
        }

        public void Motor14() throws RemoteException
        {
                ControlaMotores.controle(14,0);
        }

        public void Motor15() throws RemoteException
        {
                ControlaMotores.controle(15,0);
        }

        public void Motor16() throws RemoteException
        {
                ControlaMotores.controle(16,0);
        }

        public void Motor17() throws RemoteException
        {
                ControlaMotores.controle(17,0);
        }

        public void Motor18() throws RemoteException
        {
                ControlaMotores.controle(18,0);
        }


        public void Motor1par(int x) throws RemoteException
        {
                ControlaMotores.controle(19,x);

        }

        public void Motor2par(int x) throws RemoteException
        {
                ControlaMotores.controle(20,x);
        }

        public void Motor3par(int x) throws RemoteException
        {
                ControlaMotores.controle(21,x);
        }

        public void Motor4par(int x) throws RemoteException
        {
                ControlaMotores.controle(22,x);
        }

        public void Motor5par(int x) throws RemoteException
        {
                ControlaMotores.controle(23,x);
        }
        ……..…..
}
```

Todas as classes de servidor devem ser estendidas a classe *RemoteServer* do pacote *java.rmi.server*, mas trata-se de uma classe abstrata que define apenas os mecanismos básicos

para a comunicação entre objetos servidores e seus *stubs*[22] remotos. A classe *UnicastRemoteObject* que acompanha o RMI estende a classe abstrata *RemoteServer* e é concreta, portanto, pode-se usá-la sem escrever nenhum código [14].

Um objeto *UnicastRemoteObject* reside em um servidor, e ele deve estar ativo quando um serviço for solicitado e deve ser acessado através do protocolo TCP/IP.

Depois de criada e compilada a classe *ServidorRMI*, deve-se gerar stubs para esta. A ferramenta *rmic* as gera automaticamente com a seguinte linha de instrução:

- rmic ServidorRMI

Esta chamada gera dois arquivos de classes citados a seguir:

- ServidorRMI_Stub.class
- ServidorRMI_Skel.class

Na figura 4.9 é apresentado o diagrama da arquitetura RMI. A transmissão dos dados na rede se dá através do protocolo TCP/IP, na camada de transporte, de maneira transparente.

Figura 4.9: Diagrama da Arquitetura RMI.

Muitos sistemas operacionais, como o *Windows 2000* e o *XP* por motivos de segurança restringem o acesso às portas de I/O em nível de usuário. Como o aplicativo servidor do software de controle faz referência direta à porta paralela do computador, houve a necessidade da utilização de um aplicativo de nome *Userport*, desenvolvido por *Thomas Franzon* em 2001

[22] Classes que reúnem os parâmetros e os resultados das chamadas de métodos através da rede.

[16] e [17], que habilitou as portas necessárias para a comunicação com a porta paralela, no caso o endereço dos registradores para LPT1 é 0x37A.

4.2.4 – LOCALIZANDO OBJETOS SERVIDORES

Um programa servidor registra objetos com o serviço de registro de partida e o cliente recupera *stubs* para esses objetos. Registra-se um objeto servidor fornecendo ao serviço de registro de partida uma referência ao objeto e um nome. O nome é uma string e espera-se ser única. Na tabela 6, tem-se a implementação no lado do servidor:

Tabela 6: Definição do Objeto e do Nome do Serviço

```
String serviceName = "SCORBOT";
ServidorRMI myCount = new ServidorRMI();
Naming.rebind(serviceName, myCount);
```

onde:

- serviceName define o nome do serviço e;
- myCount faz a referência ao objeto.

O código cliente obtém uma *stub* para acessar este objeto servidor, especificando o nome do servidor e o nome do objeto, mostrado na tabela 7 da seguinte maneira:

Tabela 7: Definição do nome do servidor e o nome do objeto

```
String url ="rmi://localhost/";

InterfaceRMI myCount = (InterfaceRMI) Naming.lookup(url+"SCORBOT");
```

Na tabela 8, encontra-se o programa servidor baseado em RMI para análise.

Tabela 8: Implementação da classe *ServidorRMIServer.java*

```
import java.rmi.server.*;
import java.rmi.*;
 public class ServidorRMIServer
{       public static void main(String[] args)
        {
                try
                {
                        String serviceName = "SCORBOT";
                        ServidorRMI myCount = new ServidorRMI();
                        //Registry r = LocateRegistry.getRegistry();
                        Naming.rebind(serviceName, myCount);
                        System.out.println("Count Server ready.");
```

```
                    }
                    catch (Exception e) {
                            System.out.println("Exception: " +
e.getMessage());
                            //e.printStackTrace();
                    }
            }
    }
```

4.3 – JDBC – CONECTIVIDADE COM BANCO DE DADOS JAVA

Esse pacote permite aos programadores se conectarem com um banco de dados, consultá-lo ou atualizá-lo, usando o SQL (*Structured Query Language*, acesso à banco de dados). JAVA e JDBC possuem uma vantagem fundamental sobre outros ambientes de programação de banco de dados, pois, os programas desenvolvidos com essa linguagem de programação são independentes de plataforma e fornecedor.

O mesmo programa de banco de dados, escrito na linguagem de programação JAVA, pode ser executado em um computador NT, um servidor Solaris ou um aplicativo de banco de dados utilizado na plataforma JAVA [14]. No futuro, devido à universalidade da linguagem de programação JAVA e do JDBC, estes finalmente substituirão as linguagens de banco de dados proprietárias.

Para o desenvolvimento deste projeto, utilizou-se JDBC para a conexão com um banco de dados *Firebird*. Este é composto por apenas uma tabela chamada SCORBOT, onde a mesma possui os seguintes campos:

SCORBOT = {<u>codmotor</u>, motor, deslocamento}

Este banco de dados tem a finalidade de pré-programar tarefas a serem executadas pelo robô. A máquina cliente, que solicita a execução de uma tarefa de manipulação do robô, acessa remotamente o banco de dados no servidor e configura o deslocamento de cada motor pertencente ao braço mecânico com coordenadas específicas.

A conexão utilizando JDBC entre a aplicação e o banco de dados Firebird é feita informando o caminho de onde se encontra o banco de dados a ser utilizado. Na tabela 9, é mostrado que no programa de controle do robô criou-se variáveis do tipo *String*, que especificam o caminho, o usuário e a senha do banco de dados.

Tabela 9: Variáveis de Caminho, Usuário e Senha

```
String sUrl = "jdbc:firebirdsql:localhost:c:/APJAVA/motores.fdb";
String vUsuario = "SYSDBA";
String vSenha = "masterkey";
```

Para a conexão com o banco de dados especificado na variável sUrl, deve-se carregar o Driver JDBC específico deste banco de dados e depois abrir a conexão com o banco utilizando a função *DriverManager*, como mostrado na tabela 10.

Tabela 10: Conexão com o banco de dados

```
Class.forName("org.firebirdsql.jdbc.FBDriver");
conApJava = DriverManager.getConnection(sUrl, vUsuario, vSenha);
```

4.4 - JMF (JAVA MEDIA FRAMEWORKS)

A biblioteca JAVA Media Framework (JMF) é uma API que possui a capacidade de trabalhar com mídia baseada em tempo, como se faz em aplicações JAVA e *applets*, provendo apoio de captura e armazenamento de mídia. Para o recebimento ou envio de mídia ao vivo (tempo real) na Internet ou Intranet, precisa-se entender o recebimento e a transmissão de *streams*. JMF utiliza o Protocolo de Transporte de Tempo Real (RTP) para receber e transmitir *streams* de mídia pela rede [18].

Utilizou-se conceitos de JMF para a captura de imagens em tempo real através de uma WebCam de marca PCTronix a qual disponibiliza as imagens capturadas no formato JPEG. Com a utilização de JMF e o aplicativo JMF STUDIO da *Sun Microsystems* consegue-se detectar dispositivos de entrada e saída conectados ao computador de forma fácil e interativa,

pois este aplicativo nos mostra todas as características destes dispositivos (som e vídeo). A WebCam é conectada ao computador através da porta USB.

4.4.1 - *MEDIA STREAMING*

O termo *Media Streaming* é usado freqüentemente para definir a técnica de envio e recebimento de pacotes em cima da rede em tempo real. Quando o conteúdo de mídia é transmitido a um cliente em tempo real, este pode começar a executá-lo sem ter que esperar seu carregamento completo – transmissões ao vivo. Quando o fluxo não possui uma duração definida, seria impossível carregá-lo inteiramente.

Observa-se *Media Streaming* em muitos lugares da *Web*, rádio e televisão ao vivo, por exemplo, a transmissão de eventos que estão sendo oferecidos por um número crescente de portais de *Web*, possibilitando a administração de conferências em cima da Internet. A entrega de conteúdo de mídia dinâmica pela rede está mudando conceitos de como as pessoas se comunicam e acessam estas informações.

As imagens são transmitidas do servidor ao cliente sem que haja a interrupção para carregamento ou armazenamento das imagens. Para que se possa entender como é feita esta transmissão, alguns conceitos e definições são mostrados nos tópicos seguintes.

4.4.2 - PROTOCOLOS DE *MEDIA STREAMING*

O HTTP e protocolos de FTP estão baseados no Protocolo de Controle de Transmissão (TCP), que é um protocolo pertencente à camada de transporte projetado para comunicações de dados seguros. Se um pacote for perdido ou corrompido, será retransmitido para garantir a transferência de dados com segurança e reduzir a velocidade da taxa de transmissão global [18].

Protocolos subjacentes diferentes de TCP são tipicamente usados para transmissão de pacotes de mídia. Nesta linha, destaca-se o *User Datagram Protocol* (UDP). Este é um

protocolo incerto, não se tem garantia que a informação alcançará seu destino e que chegará na ordem em que foi enviada. O receptor se encarrega de compensar e detectar dados perdidos, pacotes duplicados e defeituosos.

O protocolo padrão da Internet para o transporte de dados em tempo real, como áudio e vídeo, é o RTP (Real Time Protocol).

4.4.3 - PROTOCOLO DE TRANSPORTE DE TEMPO REAL - RTP

O RTP provê entrega de pacotes na rede fim-a-fim para a transmissão de dados em tempo real. É independente, entretanto é freqüentemente usado com o protocolo UDP.

A figura 4.10 ilustra a arquitetura do Protocolo RTP.

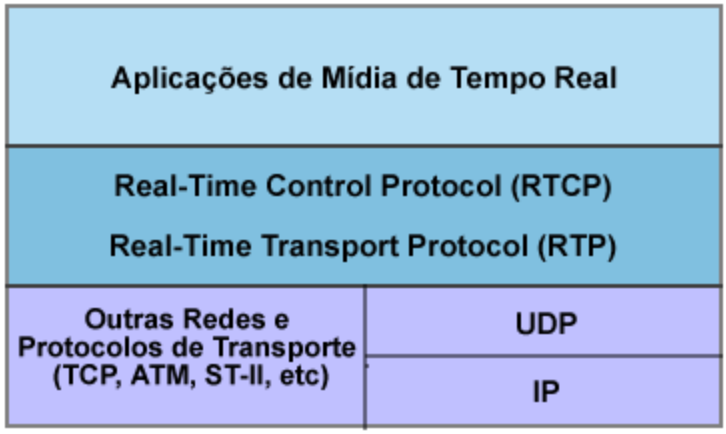


Figura 4.10: Arquitetura RTP

O RTP pode ser usado em serviços de rede *unicast* (difusão) e *multicast* (multidifusão). No serviço de rede *unicast*, são enviadas cópias separadas dos dados da fonte para cada destino. Já, em um serviço de rede de *multicast*, os dados são enviados somente uma vez da fonte e a rede é responsável para transmití-los a locais múltiplos. *Multicasting* possui maior eficiência para muitas aplicações de multimídia, como vídeo conferências. O Protocolo padrão da Internet (IP – Internet Protocol) apóia *multicasting*. Os serviços de *multicasting* não são abordados neste trabalho de mestrado.

4.4.4 - SERVIÇOS RTP

O protocolo RTP permite identificar o tipo de dados que serão transmitidos, determina também, a ordem de apresentação dos pacotes de dados e sincroniza os *streams* de mídia de fontes diferentes [18].

Este protocolo não garante a chegada dos pacotes na ordem que eles foram enviados. Na verdade, não é garantida a chegada dos dados em seu destino. O receptor é quem se encarrega de reconstruir a sucessão de pacotes do remetente, e assim, descobrir quais foram perdidos, utilizando-se da informação contida no cabeçalho.

Enquanto o RTP não pode prover nenhum mecanismo para assegurar a entrega oportuna de pacotes ou garantir a qualidade de serviço, este controle é feito por um protocolo auxiliar denominado RTCP (*Real Time Control Protocol*), mostrado na figura 4.5, que permite monitorar a qualidade da distribuição de dados. O RTCP também provê controle e mecanismos de identificação para transmissões de RTP.

Se a qualidade de serviço for essencial para uma aplicação particular, o RTP pode ser usado em cima de um protocolo de reserva que provê serviços orientados à conexão.

4.4.5 - ARQUITETURA DE RTP

Uma sessão de RTP é uma associação entre aplicações que se comunica entre si, é identificada por um endereço de rede e um par de portas, sendo uma usada para os dados de mídia e a outra usada para controle de dados (RTCP).

Um participante é uma única máquina, ou usuário que participa da sessão. Participação em uma sessão consiste em recepção passiva de dados (o receptor), transmissão ativa de dados (remetente) ou ambos.

Cada tipo de mídia é transmitido em uma sessão diferente. Por exemplo, se áudio e vídeo são usados em uma conferência, uma sessão é usada para transmitir os dados de áudio e

outra para transmitir os dados de vídeo. Isto permite que os participantes escolham quais tipos de mídia querem receber.

Os dados de mídia para uma sessão são transmitidos em série de pacotes que por sua vez, originam de uma fonte particular. Esses pacotes são chamados fluxo de RTP. Cada fluxo contém duas partes, um cabeçalho estruturado e os dados atuais (a carga útil do pacote) [18].

4.4.6 - APLICAÇÕES DE RTP

As aplicações são divididas freqüentemente na necessidade de recepção e transmissão de dados da rede (cliente/servidor). Algumas aplicações de conferências capturam e transmitem dados no mesmo instante em que são receptores na rede. Neste trabalho, o servidor captura imagens (*streams*) da manipulação do braço mecânico e as envia para um cliente específico na rede. Para o envio dos pacotes de informações, o servidor especifica para quem será enviado os *frames* através de um endereço IP, e só a máquina cliente destino poderá ter acesso a estas informações – *unicasting.*

4.4.7 - TRANSMISSÃO E RECEPÇÃO DE STREAMS RTP

Na criação de uma transmissão de mídia ao vivo, o servidor utilizado neste trabalho tem a seguinte configuração:

- ✓ O *DataSource* é um *CaptureDevice* – Dispositivo de captura de som ou imagem (microfone ou webcam), para o trabalho utilizou-se uma câmera digital PCTronix do tipo "vfw:Microsoft WDM Image Capture (Win32):0";

Para a definição do dispositivo de captura de imagens, foi criada a classe *SwingCapture.class.* O código fonte esta na tabela 11.

Tabela 11: Código fonte do arquivo *SwingCapture.java*

```
public class SwingCapture extends Panel
{
        public static Player player = null;
        public CaptureDeviceInfo di = null;
        public MediaLocator ml = null;
```

```
        public JButton capture = null;
        public Buffer buf = null;
        public Image img = null;
        public VideoFormat vf = null;
        public BufferToImage btoi = null;
         public SwingCapture()
        {
                setLayout(new BorderLayout());
                setSize(800,600);

                String str2 = "vfw:Microsoft WDM Image Capture (Win32):0";
                di = CaptureDeviceManager.getDevice(str2);
                ml = di.getLocator();
                player = Manager.createRealizedPlayer(ml);
                player.start();
                Component comp;

                if ((comp = player.getVisualComponent()) != null)
                    {
                     add(comp,BorderLayout.CENTER);
                     }
        }
}
```

- ✓ O *Processor* foi implementado para ler os dados do dispositivo de captura (WebCam). Os dados capturados são formatados em um padrão específico, caso o dispositivo já não forneça um;

A codificação para a criação e formatação dos dados capturados é ilustrada na tabela 12 pela função *createProcessor*().

Tabela 12: Função *createProcessor*

```
private String createProcessor()
{
      DataSource ds = null;
      processor = inProc;

      ds = Manager.createDataSource(locator);

      // Aguarde a configuração do Processor
      boolean result = waitForState(processor, Processor.Configured);

      // Obtém trilhas do Processor
      TrackControl [] tracks = processor.getTrackControls();

      boolean programmed = false;

      // Realiza a procura de trilhas de vídeo
          for (int i = 0; i < tracks.length; i++)
          {
              Format format = tracks[i].getFormat();
              if (     tracks[i].isEnabled() && format instanceof VideoFormat &&
!programmed)
              {
                      // Faz a procura de trilhas de vídeo e tenta converter para o formato
```

```
JPEG/RTP
                    // Verifica se o tamanho dos pacotes são múltiplos de 8

                    Dimension size = ((VideoFormat)format).getSize();
                    float frameRate = ((VideoFormat)format).getFrameRate();

                    int w = (size.width % 8 == 0 ? size.width :
                                    (int)(size.width / 8) * 8);
                    int h = (size.height % 8 == 0 ? size.height :
                                    (int)(size.height / 8) * 8);
                    VideoFormat jpegFormat = new VideoFormat(VideoFormat.JPEG_RTP,
                                                        new Dimension(w, h),
                                                        Format.NOT_SPECIFIED,
                                                        Format.byteArray,
                                                        frameRate);
                            tracks[i].setFormat(jpegFormat);
                    programmed = true;
                }
            }
          ContentDescriptor cd = new ContentDescriptor(ContentDescriptor.RAW_RTP);
          processor.setContentDescriptor(cd);

          result = waitForState(processor, Controller.Realized);
          setJPEGQuality(processor, quality/100);

          // Adquire os dados do Processor
          dataOutput = processor.getDataOutput();
          return null;     }
```

✓ O *Render* é um *SendStream* responsável pelo envio dos dados capturados pela internet, via RTP.

Para o envio dos dados capturados, foi codificado a função createTransmitter() ilustrada na tabela 13. Os dados são enviados para um endereço IP específico em tempo real.

Tabela 13: Função *createTransmitter*

```
private String createTransmitter()
 {
        // Define o endereço e a porta de destino e qual o tipos dos dados

         String rtpURL = "rtp://" + ipAddress + ":" + port + "/video";
          MediaLocator outputLocator = new MediaLocator(rtpURL);

          rtptransmitter = Manager.createDataSink(dataOutput, outputLocator);
          rtptransmitter.open();
          rtptransmitter.start();
          dataOutput.start();
          return null;
   }
```

A figura 4.11 ilustra a configuração dos passos definidos anteriormente.

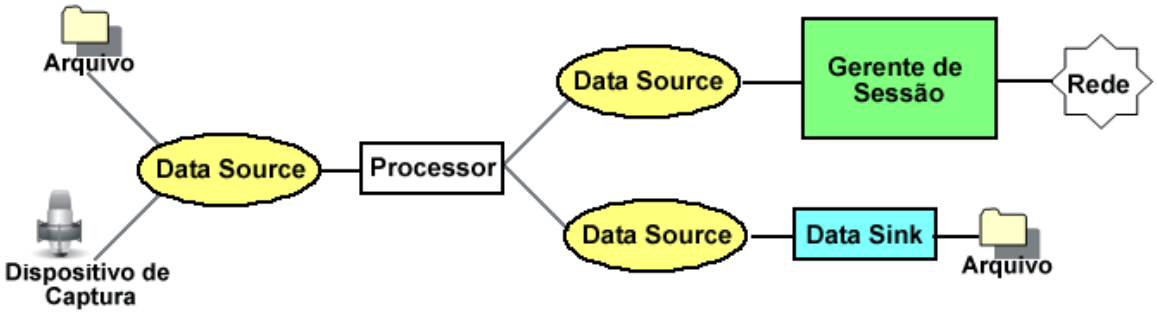


Figura 4.11: Transmissão RTP

Já na recepção da mídia em tempo real por parte do cliente, a configuração é definida a seguir.

- ✓ O *DataSource* é um *InputStream*, via RTP, que captura os dados transmitidos pelo servidor;
- ✓ O *Processor* um decodificador deste *DataSource* (Player);
- ✓ O *Render* apresenta (Player) os dados capturados em uma *applet*.

A figura 4.12 evidencia a recepção de um pacote RTP.

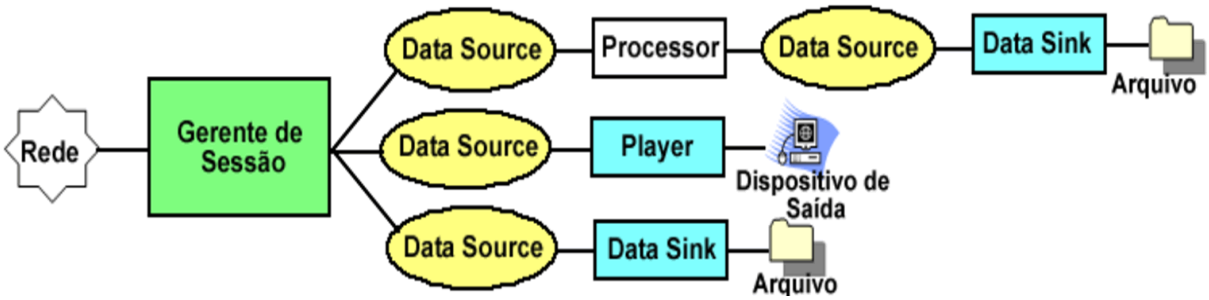


Figura 4.12: Recepção RTP

Ringgenberg [15], utilizou para o envio e recebimento de pacotes de som e imagem, artifícios como a implementação de sub-programas escritos em linguagens como C++, Cobol, Fortran, etc.; e acessadas utilizando o conceito de métodos nativos (JNI), esta forma de acesso e controle, acaba por diminuir a portabilidade dos aplicativos implementados, outros autores [2] e [13], utilizam *script's* feitos em CGI, para fazer o monitoramento e controle do braço. Assim, o software implementado neste trabalho de mestrado, em relação à portabilidade,

obteve uma crescente evolução em relação a trabalhos anteriores com a utilização de JMF/RTP.

CAPÍTULO 5 – MODELAGEM DO SOFTWARE DE CONTROLE

Neste capítulo descreve-se os conceitos que foram utilizados para o desenvolvimento deste software, no que diz respeito à modelagem orientada a objetos.

5.1 – UML

O significado de UML (*Unified Modeling Language*) é linguagem de modelagem unificada, derivada da unificação de três linguagens de modelagens, as quais são: "OMT" de Rumbauch, "o método de Booch" e os "casos de uso" de Jacobson. Cada uma destas linguagens são bem sucedidas como métodos de análise orientada a objetos, mas a UML tem como objetivo tirar proveito das principais características de cada uma.

A linguagem UML é usada para transmitir e fornecer algumas informações detalhadas de como um sistema deve ser implementado para efetuar o que o usuário solicita de forma satisfatória. Cada conceito a ser modelado tem uma representação gráfica específica na linguagem. É importante seguir as convenções definidas na UML para que outras pessoas possam entender os diagramas gerados [19].

A UML é composta de 9 diagramas, mas defini-se aqui somente os diagramas de classe, objeto e caso de uso [19].

- ✓ Classe: mostra as classes que compõem o sistema e as relações entre elas. Trata-se de um aspecto estático e estrutural do sistema;
- ✓ Objeto: mostra alguns objetos (instâncias das classes) e as relações entre eles. Os objetos supõem a execução do programa e que instâncias são criadas para enviar e receber mensagens;
- ✓ Caso de Uso: mostra como o sistema vai interagir com o usuário.

5.2 – DIAGRAMA DE CLASSE

O diagrama de classe lista todos os conceitos do domínio que serão implementados no sistema. Por exemplo, em um domínio de reserva de lugares, os itens importantes seriam: lugar, data e cliente. O diagrama mostra também as relações entre os conceitos: um cliente "reserva" *x* lugar(es) para uma data específica. Normalmente, conceitos são tratados como substantivos e relações como verbos.

O diagrama de classes define a estrutura do sistema a ser desenvolvido.

Em UML as classes são representadas por caixas divididas em três partes: nome da classe, atributos e operações como mostrado na figura 5.1.

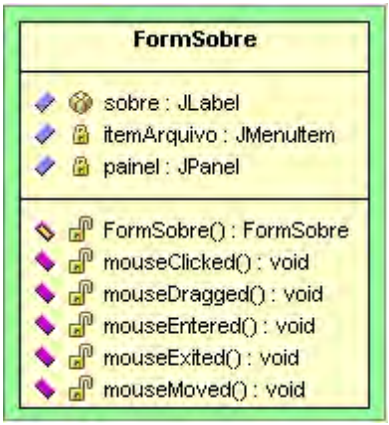


Figura 5.1 – Representação de uma classe usando UML

Na figura 5.1, o nome da classe é FormSobre, os atributos são identificados pelos nomes: sobre, itemArquivo e painel respectivamente; já as operações são definidos pelos itens expressos na linha 3 do referido diagrama.

Um atributo representa uma propriedade que todos os objetos da classe possuem como: altura da mesa, número de pernas e posição na sala. Cada objeto possui valores particulares para seus atributos, algumas mesas são mais baixas e outras são mais altas.

Operações ou métodos definem uma função que os objetos devem efetuar. Por exemplo, abrir, fechar ou deslocar uma janela são operações de uma determinada classe.

5.3 – DIAGRAMA DE OBJETOS

O diagrama de objetos é uma variação do diagrama de classes e utiliza quase a mesma notação. A diferença é que o diagrama de objetos mostra os objetos que foram instanciados nas classes. O diagrama de objetos é como se fosse o perfil do sistema em um certo momento de sua execução.

A mesma notação do diagrama de classes é utilizada com duas exceções: os objetos são escritos com seus nomes sublinhados e todas as instâncias em um relacionamento são mostradas. Os diagramas de objetos não são tão importantes como os diagramas de classes, mas eles são muito úteis para exemplificar diagramas complexos de classes ajudando muito em sua compreensão. Diagramas de objetos também são usados como parte dos diagramas de colaboração, onde a colaboração dinâmica entre os objetos do sistema são mostrados [20].

5.4 – DIAGRAMA DE CASO DE USO

Um diagrama de caso de uso (ou *use case*) é um tipo de classificador representando uma unidade funcional coerente provida pelo sistema, subsistema, ou classe manifestada por seqüências de mensagens intercambiáveis entre os sistemas e um ou mais atores.

Os casos de uso foram propostos inicialmente por Ivar Jacobson em sua metodologia de desenvolvimento de sistemas orientados a objetos. Posteriormente foi incorporado à UML tornando seu uso uma prática freqüente na identificação de requisitos de um sistema. Nesse contexto um caso de uso descreve um comportamento que o software a ser desenvolvido apresentará quando estiver pronto.

Um software freqüentemente é um produto complexo, e sua descrição envolve a identificação e documentação de vários casos de uso, cada um deles descrevendo uma parcela da que o software ou uma de suas partes deverá oferecer.

É importante notar que os casos de uso não descrevem como o software deverá ser construído, e sim, como ele deverá se comportar.

A UML define um diagrama para representar graficamente os casos de uso e seus relacionamentos.

Nas próximas sessões somente os diagramas de classes serão tratados para o software de controle do SCORBOT ER - III.

5.5 – DIAGRAMAS DE CLASSES DO APLICATIVO SERVIDOR

Neste tópico são mostrados os diagramas de classes que compõem o aplicativo servidor e por fim o relacionamento entre as classes. A figura 5.2, mostra a classe principal do aplicativo servidor *(ServidorRMIServer*) com a definição de seus atributos e suas operações. A figura 5.2 também indica os métodos e variáveis utilizadas na implementação da classe *ServidorRMIServer*.

As setas tracejadas da figura 5.2 representam as dependências entre as classes. A relação de dependência entre as classes é representada pelo tracejado iniciado na classe que depende para a classe sobre a qual ela depende. Já as generalizações indicam que a superclasse *Object* do pacote *java.lang* possui uma sub-classe que é a classe ServidorRMIServer. O funcionamento da subclasse é dado como se a superclasse estivesse sido copiada integralmente para a subclasse. Estas definições também são expressas nos próximos diagramas.

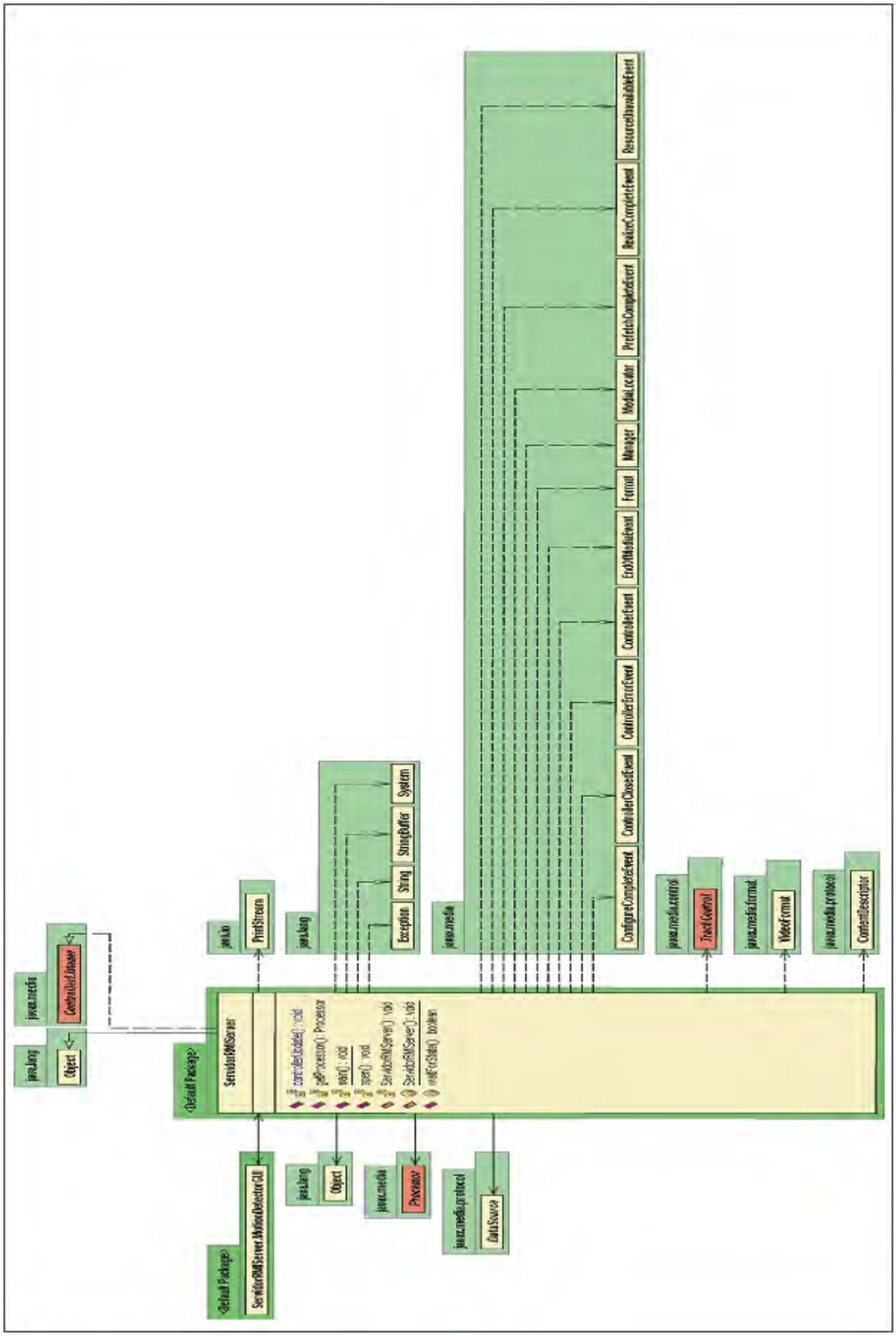


Figura 5.2: Diagrama da Classe *ServidorRMIServer* e recursos usados da API JAVA.

Na figura 5.3 é mostrado o diagrama da classe ServidorRMI que efetivamente executa os métodos anunciados na classe de interface remota. A subclasse *ServidorRMI* herda o comportamento da superclasse *UnicastRemoteObject*, pertencente ao pacote *java.rmi.server*, e faz a utilização dos recursos disponibilizados pelos métodos e variáveis da classe *RemoteException*.

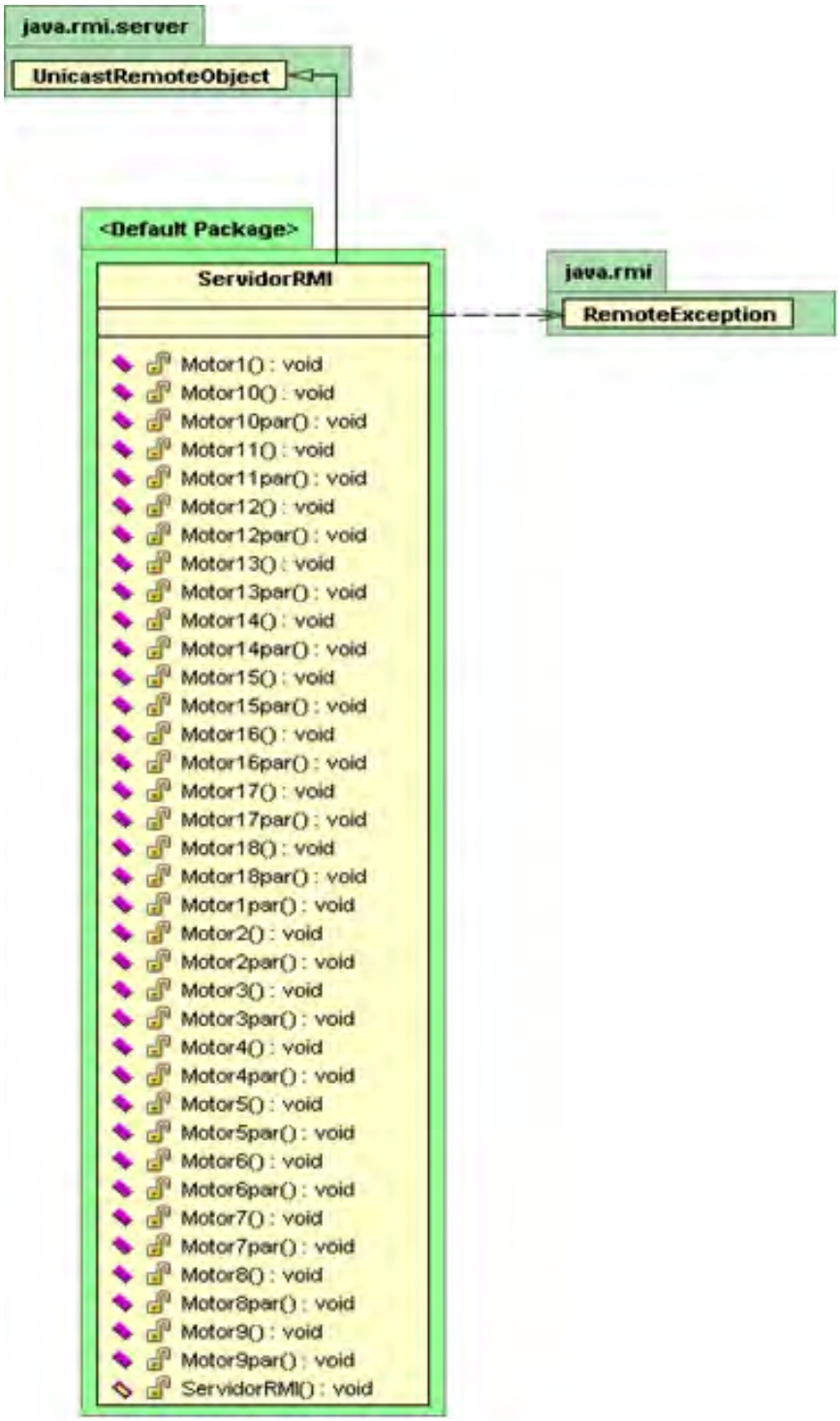


Figura 5.3: Diagrama da Classe *ServidorRMI* e recursos usados da API JAVA

Na figura 5.4 é mostrado o diagrama da classe *InterfaceRMI*, que executa a comunicação entre o cliente e o servidor na manipulação do braço mecânico. Esta classe herda as características da classe *Remote*, pertencente ao pacote *RMI* e também, faz o uso dos métodos e variáveis da classe *RemoteException.*

Figura 5.4: Diagrama da Classe *InterfaceRMI* e recursos usados da API JAVA

A figura 5.5 mostra a que a classe *ControlaMotores* herda o comportamento da classe *Object* do pacote *lang*,e faz uso dos recursos oferecidos por métodos e variáveis das classes *String* e *System*.

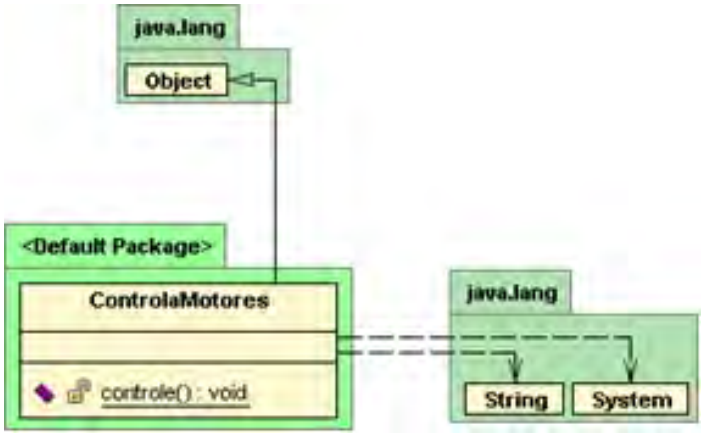


Figura 5.5: Diagrama da Classe *ControlaMotores* e recursos usados da API JAVA

Na figura 5.6 é mostrado o diagrama da classe SwingCapture, que se encarrega da aquisição das imagens através da webcam e a formatação dos dados em seu formato específico. A classe *SwingCapture* utiliza-se das características da classe *Panel* e faz a utilização dos recursos dos métodos e atributos das classes restantes do diagrama.

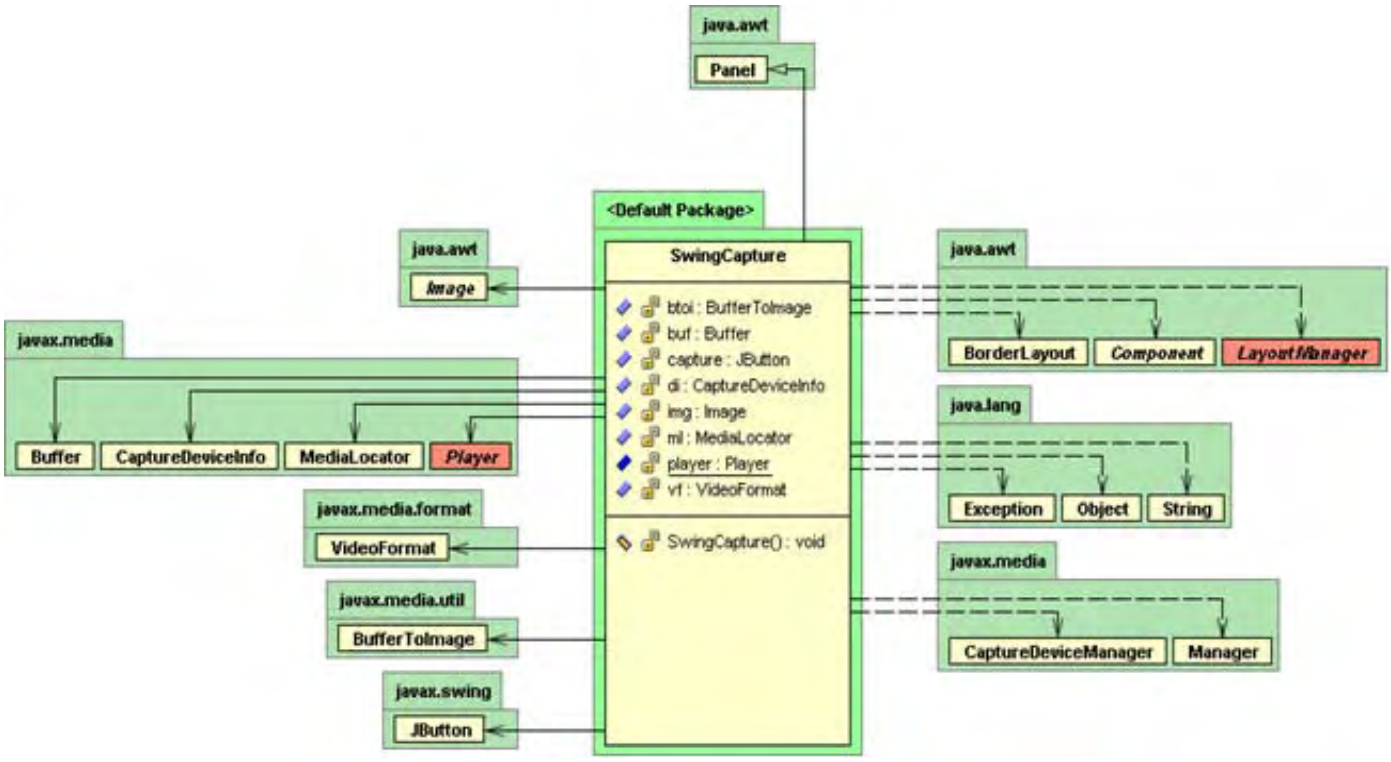


Figura 5.6: Diagrama da Classe *SwingCapture* e recursos usados da API JAVA

O diagrama da classe Clone, responsável pela cópia dos *frames* capturados pelo dispositivo de vídeo é ilustrado na figura 5.7.

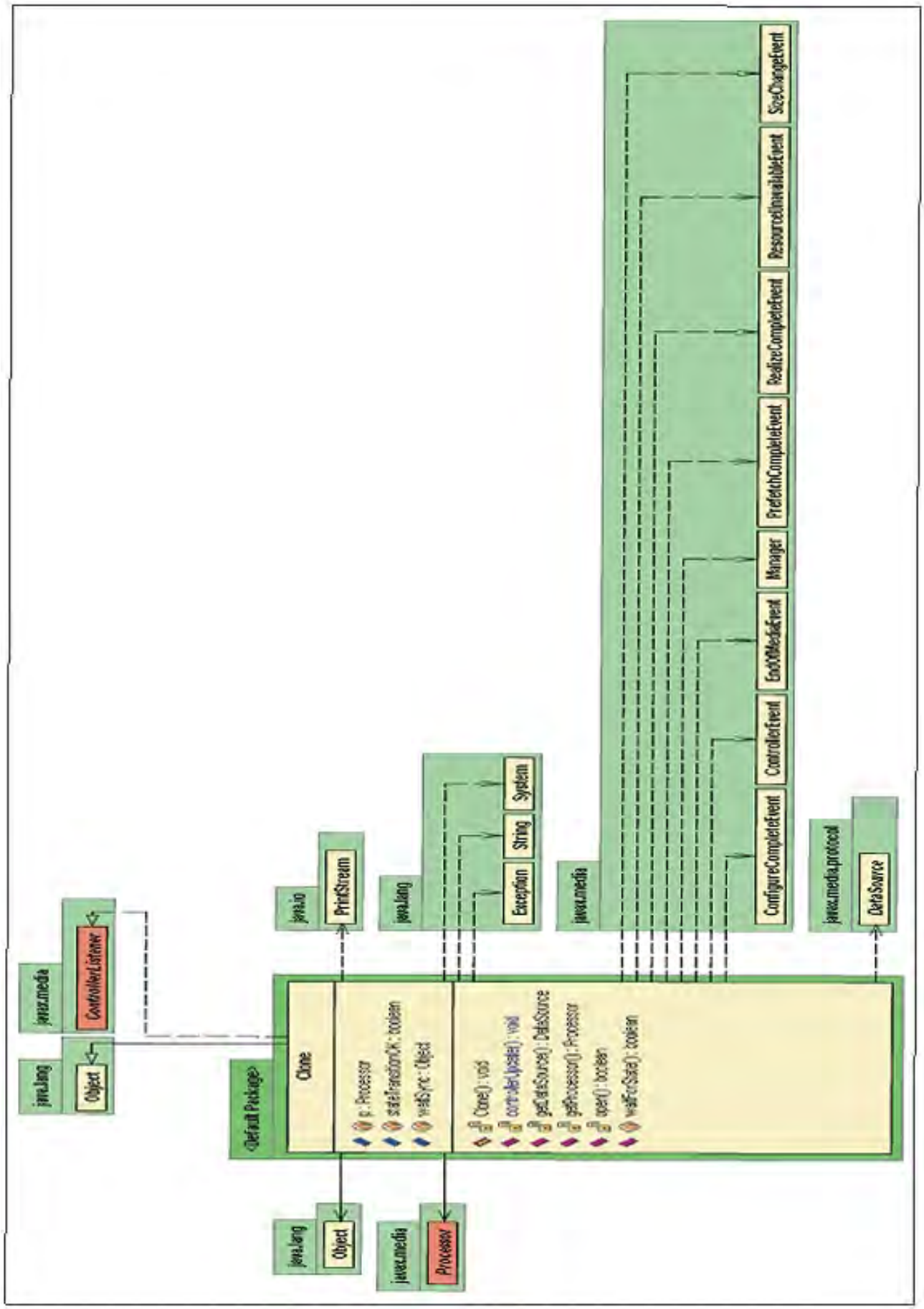


Figura 5.7: Diagrama de Classe Clone e recursos usados da API JAVA

O diagrama 5.8 mostra a classe *TransmiteVideo*, responsável pelo envio dos quadros capturados através de *broadcasting* para a rede de computadores.

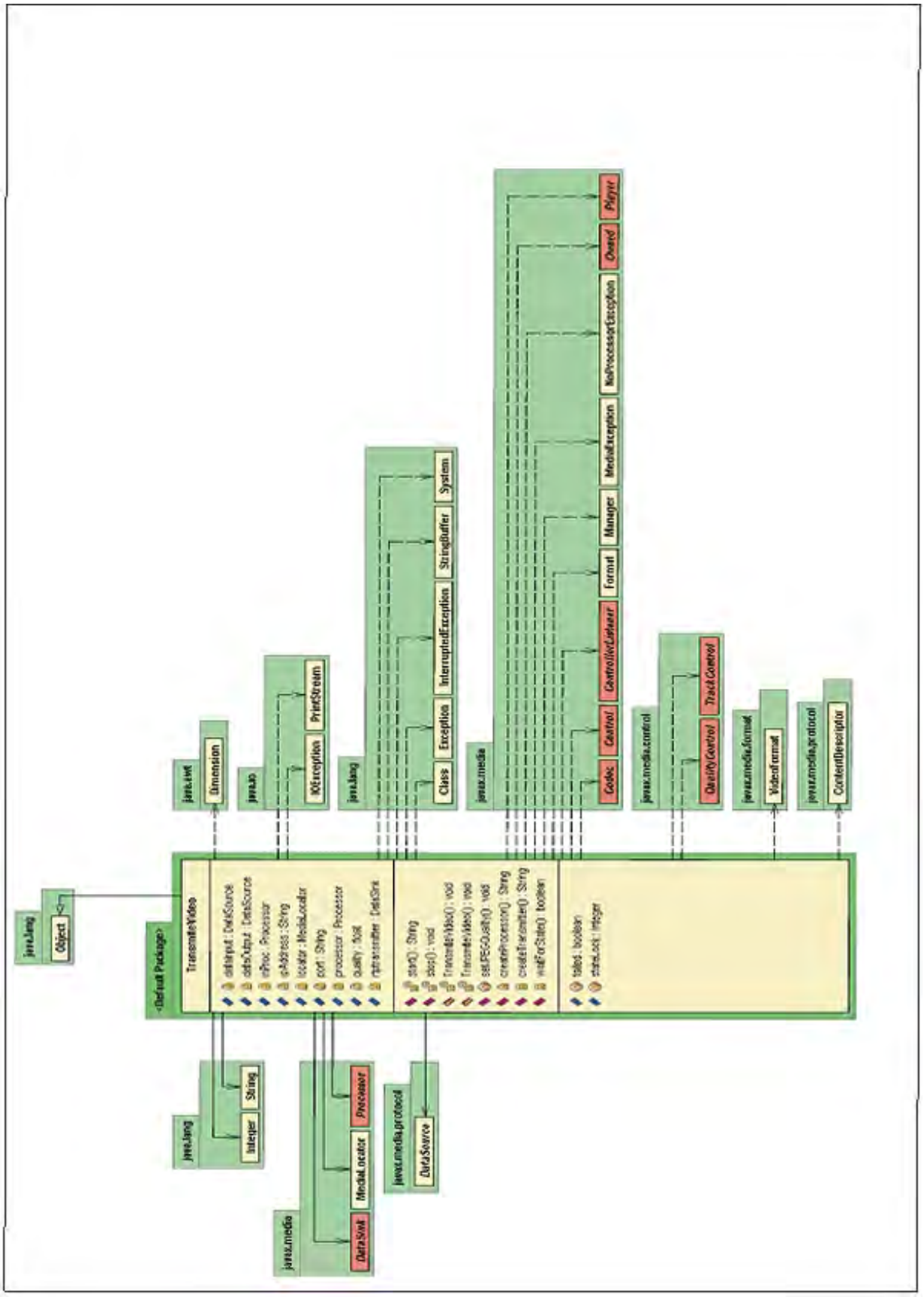


Figura 5.8: Diagrama de Classe TransmiteVideo e recursos usados da API JAVA

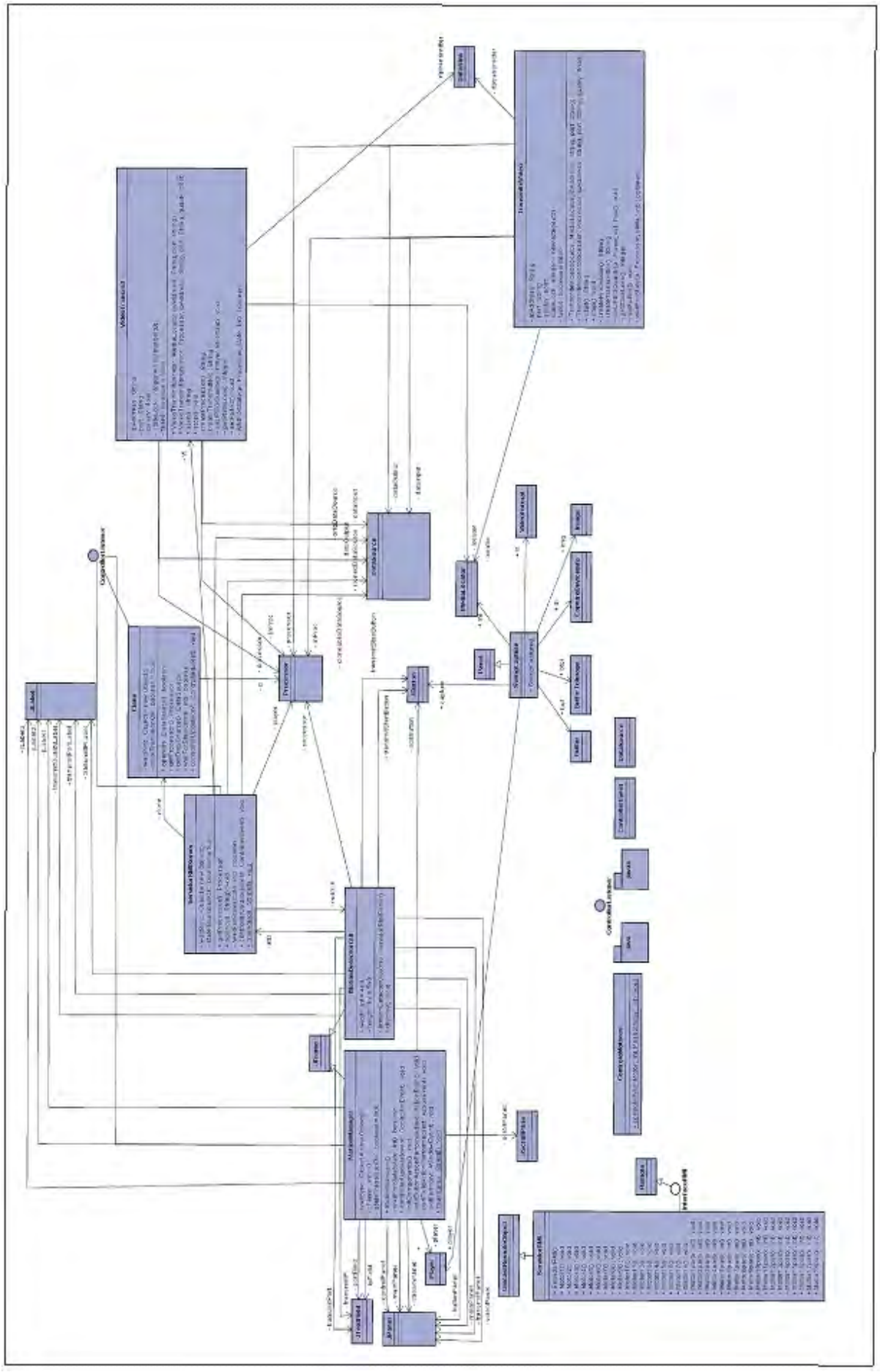


Figura 5.9: Diagrama completo do Aplicativo Servidor.

A figura 5.9 mostra o diagrama de classe com o relacionamento entre as classes do aplicativo servidor.

5.6 – DIAGRAMAS DE CLASSES DO APLICATIVO CLIENTE

Neste tópico são mostrados os diagramas de classes que compõem o aplicativo cliente por fim o relacionamento entre as classes.

A figura 5.10 mostra o diagrama da classe *AbrePrograma*, que chama o formulário de principal do software de controle do robô. Este diagrama indica a existência de uma operação chamada *AbrePrograma()*.

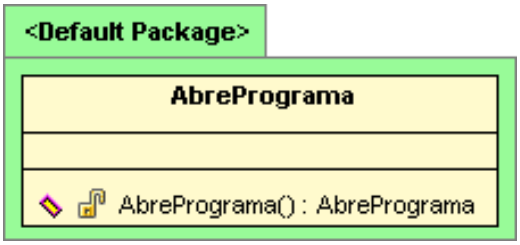


Figura 5.10: Diagrama de Classe *AbrePrograma*.

Na figura 5.11 é mostrado o diagrama da classe *classeProcesso*, responsável pelo controle de múltiplas linhas de execução (*trheads*), para evitar que o aplicativo não pare de responder a solicitações do usuário. Neste diagrama podemos observar as operações *classeProcesso(), run(), readObject()* e *writeObject()*. Todas estas operações se encarregam do controle da execução de múltiplas linhas de programação.

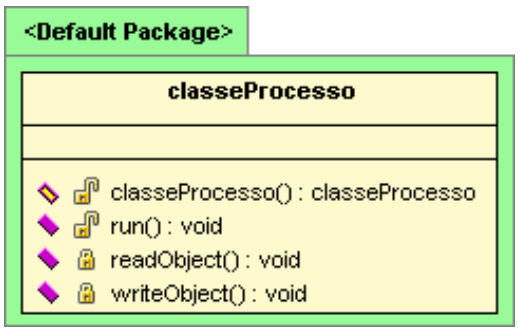


Figura 5.11: Diagrama de Classe *ClasseProcesso*.

Na figura 5.12 é mostrado o diagrama *InterfaceRMI*, que se encontra do lado da máquina cliente e já definido na figura 5.4.

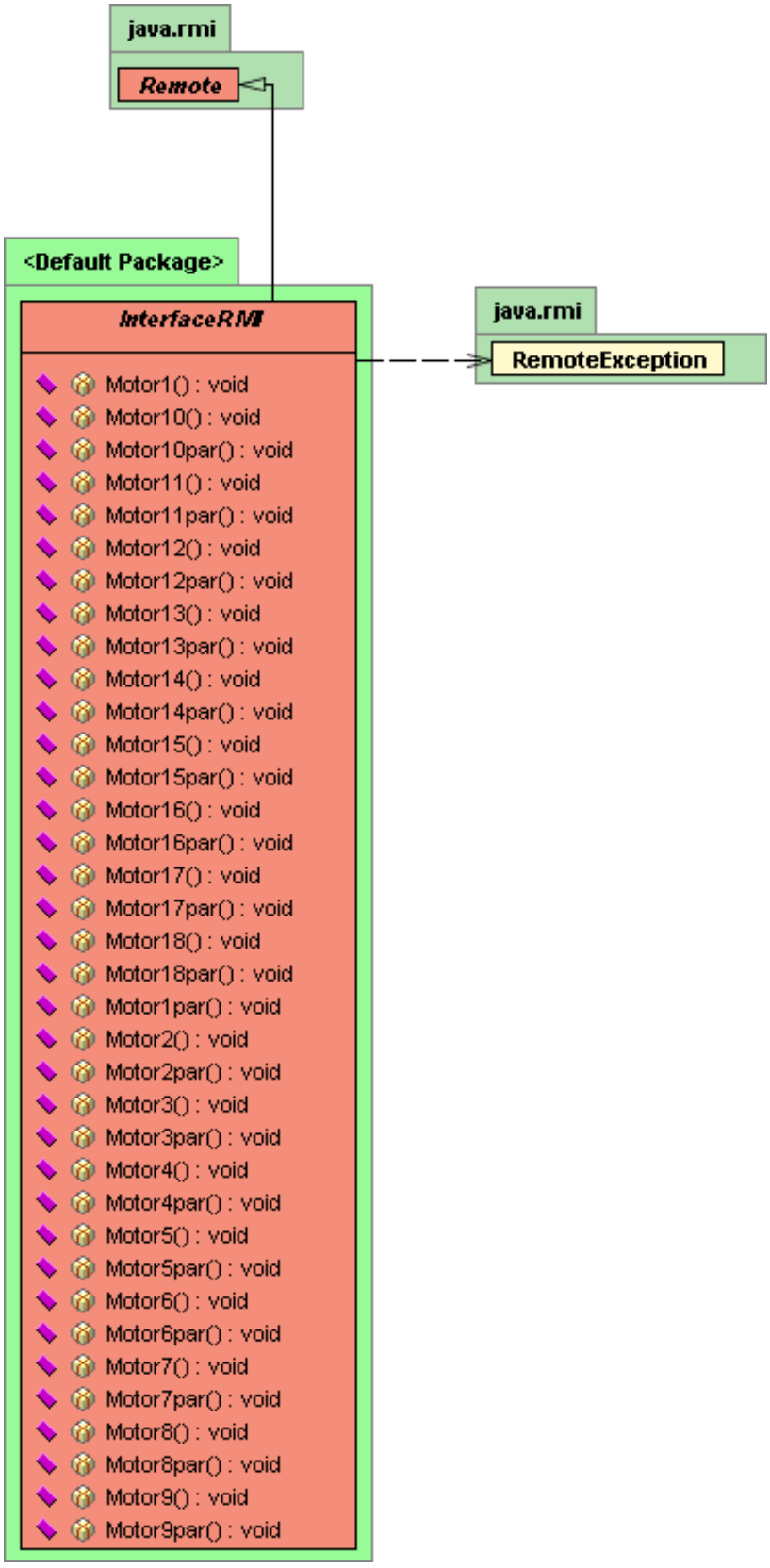


Figura 5.12: Diagrama de Classe InterfaceRMI do aplicativo cliente.

N figura 5.13 mostra-se o diagrama da classe *FormSobre*, classe responsável pela criação do formulário que traz as especificações da criação do software de controle. A classe *FormSobre* é definida com seus objetos e suas operações, como mostra o diagrama.

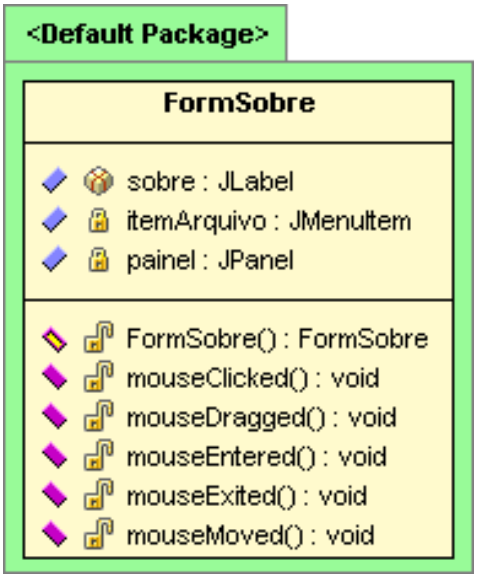


Figura 5.13: Diagrama de Classe FormSobre.

Na figura 5.14 mostra-se o diagrama da classe *FormBanco*, responsável pelo cadastro informações em um banco de dados para a automação do robô. Esta classe herda o comportamento da classe *JFrame* do pacote *Swing*.

Na figura 5.15 é mostrado o diagrama de classe completo do aplicativo cliente, neste diagrama observamos todas as dependências entre as classes participantes.

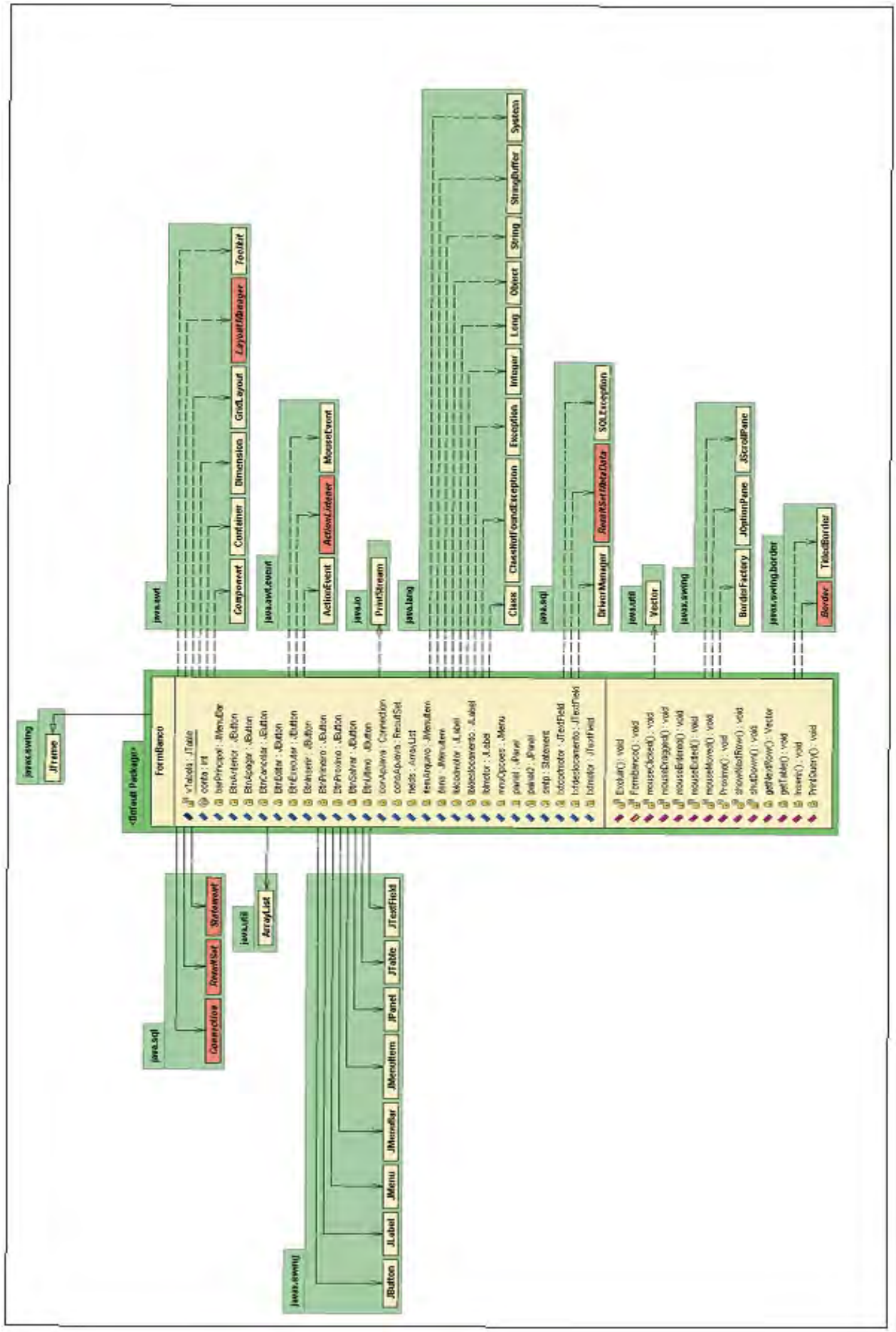


Figura 5.14: Diagrama de Classe FormBanco.

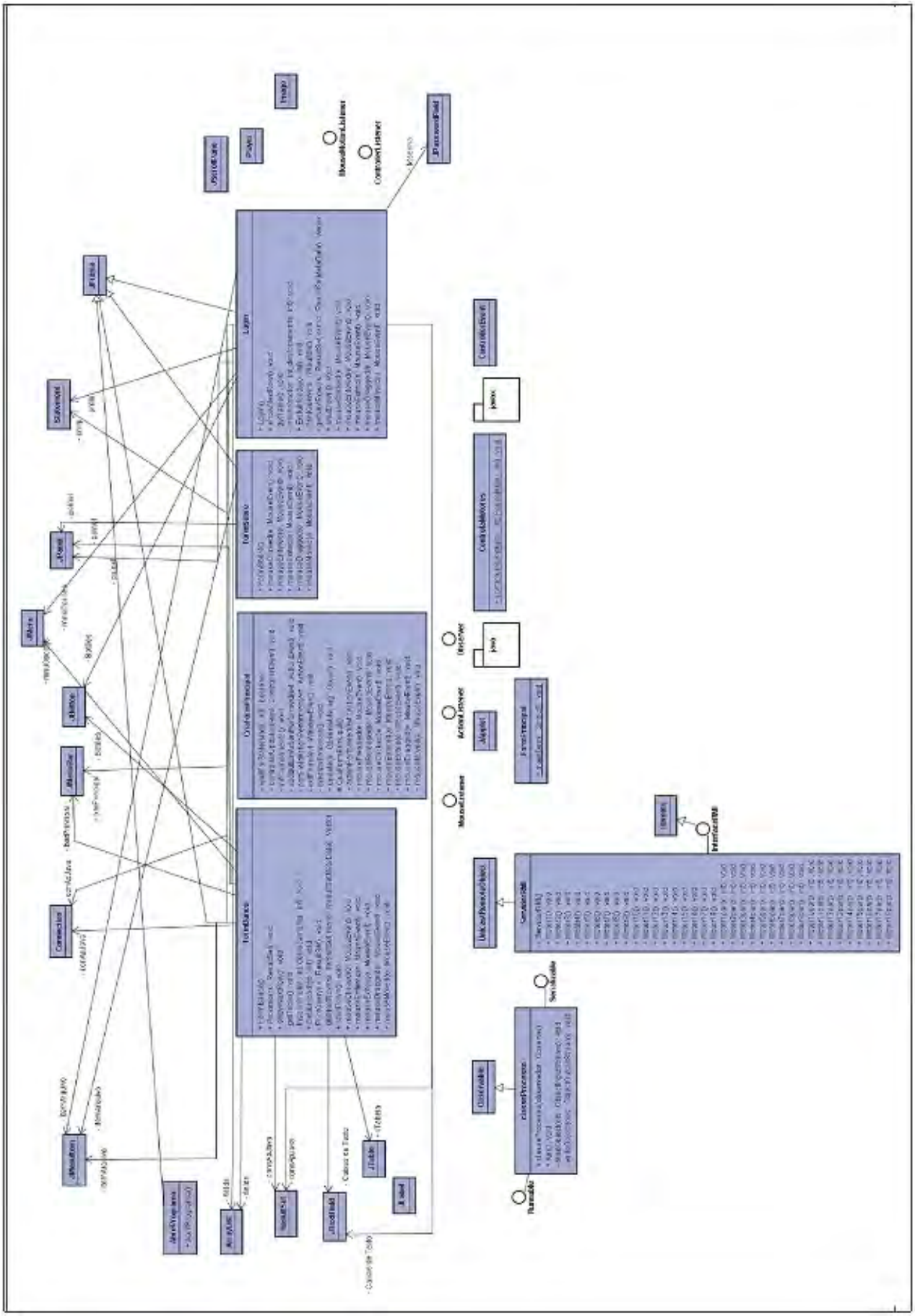


Figura 5.15: Diagrama de Classe Completo do Aplicativo Cliente

CAPÍTULO 6 – TESTES DE COMUNICAÇÃO DO SOFTWARE

Neste capítulo destaca-se o tempo de comunicação entre cliente e servidor na ativação dos motores do robô SCORBOT ER –III e também o monitoramento do tráfego na rede na transmissão dos pacotes de dados.

6.1 – TEMPO DE ATIVAÇÃO DOS MOTORES NA REDE

Os testes para a comunicação entre cliente e servidor foram efetuados a partir de uma função disponível em JAVA (*System.currentTimeMillis()*), o qual determinou o tempo efetivo global e individual de comunicação entre cliente/servidor na ativação dos motores em milissegundos.

A figura 6.1 mostra o tempo médio individual de ativação dos motores do braço mecânico. Vale ressaltar que as informações ilustradas na figura 6.1, são oriundas da execução da função de *Reset*, responsável por inicializar e posicionar as partes componentes do robô de forma adequada. As informações foram coletadas na rede local da Faculdade de Engenharia de Ilha Solteira, onde houve a comunicação de máquinas cliente/servidor entre os computadores dos laboratórios de Circuitos Digitais - LPSSD (cliente) e o Laboratório de Controle (servidor).

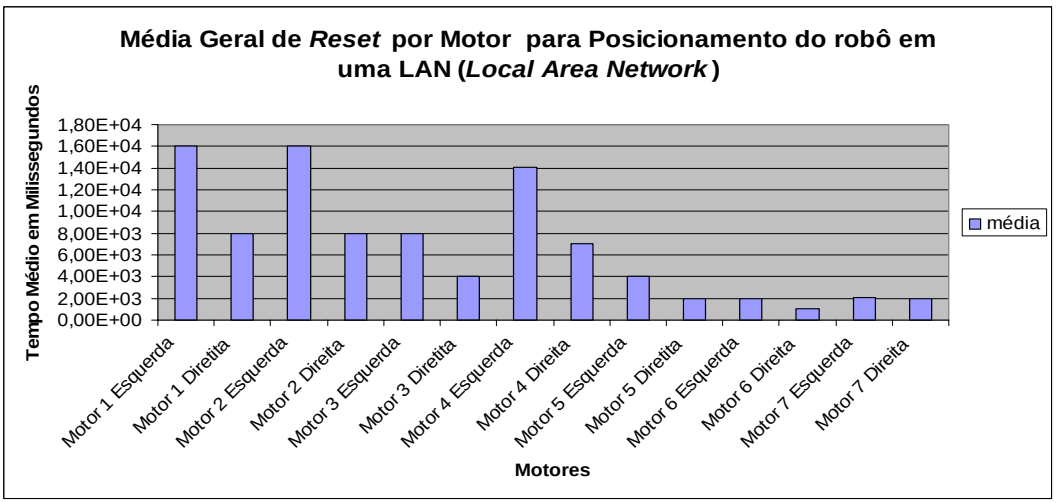


Figura 6.1: Tempo médio individual de ativação dos motores do braço mecânico – LAN.

Na figura 6.2 é mostrado o gráfico que indica o tempo global para realização da tarefa de *Reset*. Para a criação da ilustração, foi efetuado o processo de reinicialização do braço afim, de observar a variação entre um processo e outro. Como é observada, a figura 6.2 indica que o tempo global de posicionamento inicial fica em torno de 1 minuto e 34 segundos, com variações desprezíveis de forma geral.

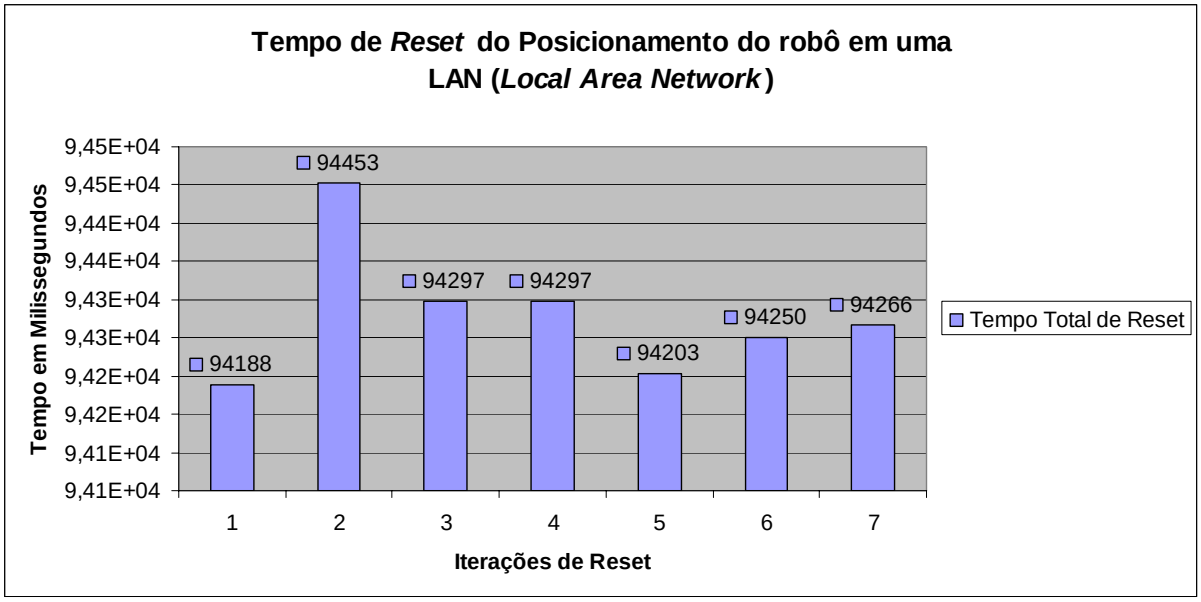


Figura 6.2: Tempo global para realização da tarefa de Reset.

6.2 – TRÁFEGO DE PACOTES NA REDE

Como já mencionado nos capítulos anteriores, a comunicação entre cliente e servidor é realizada de duas maneiras:

- ✓ Utilização de conceitos RMI, para a comunicação com o servidor remoto e;
- ✓ Conceitos de *mídia streaming* para envio dos quadros de som e imagem para a máquina participante na comunicação e controle do robô.

Devido ao alto índice de fluxo na rede em relação à transferência de pacotes, utilizando os protocolos TCP e UDP, procurou-se analisar os parâmetros na transmissão desses dados.

N figura 6.3, é apresentado um gráfico adquirido através do Gerenciador de Tarefas do sistema operacional *Windows XP*, onde é relatada a taxa de transferência de quadros de vídeos RTP na rede utilizada.

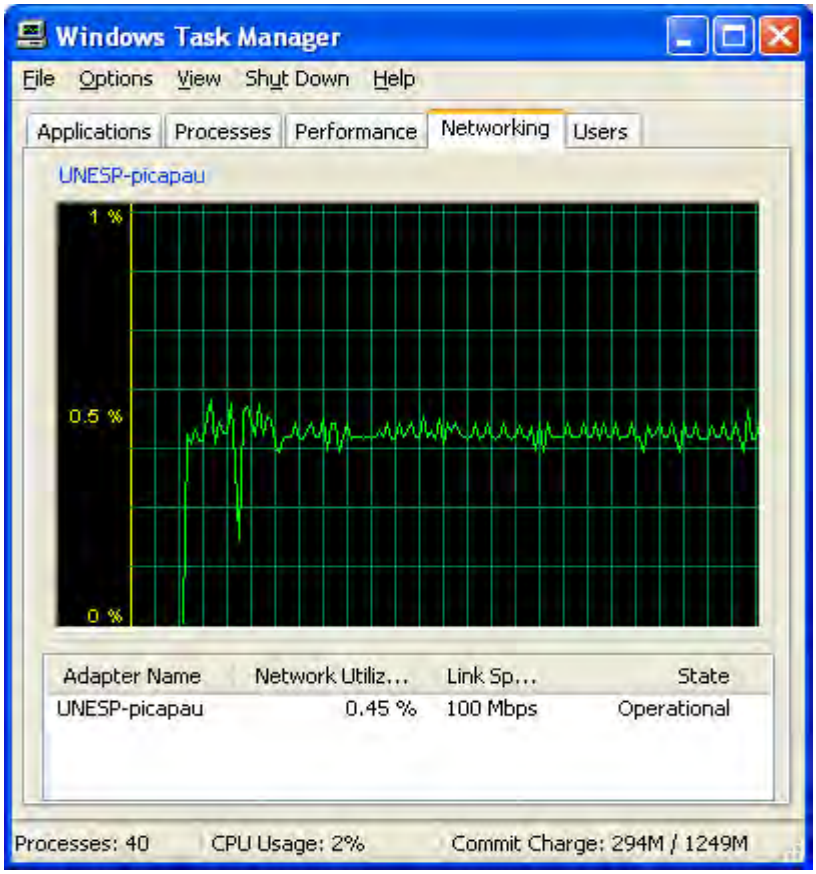


Figura 6.3: Utilização da Rede com Transmissão Vídea.

Como observado, esta rede possui uma taxa de comunicação de 100 Mbps, e durante a execução da aplicação de controle na transmissão das imagens capturadas pelo servidor à máquina cliente, somente 0.45% desta banda foi utilizada. Em termos práticos, a rede operou com uma taxa real de 460,8 Kbps.

Já a figura 6.4, traz em seu bojo, informações coletadas a partir da execução do Gerenciador de Tarefas, quando a rede operava com a transmissão de pacotes de imagens

(UDP) capturados pela webcam do servidor e enviadas ao cliente específico e, também pacotes utilizando conceitos TCP/RMI, pacotes enviados do cliente ao servidor com solicitação de execução para movimentação de partes específicas do braço.

Verifica-se que a taxa de utilização do canal de comunicação de 100 Mbps, com o envio tanto de pacotes de vídeo RTP como de troca de mensagens RMI, ficou restrito a média de 0,83% de sua capacidade total, sendo assim, a taxa média de comunicação foi de 849,92 Kbps.

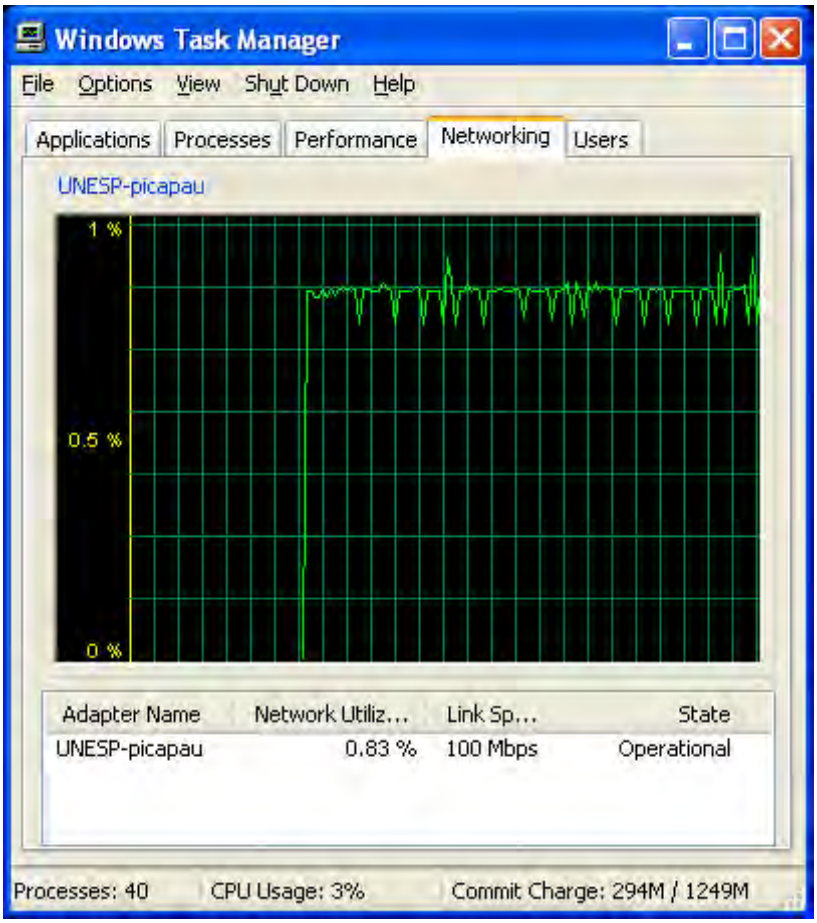


Figura 6.4: Utilização da Rede com Transmissão RTP e RMI

As análises esboçadas pelas figuras 6.3 e 6.4 foram melhores evidenciadas através de um software chamado *Ethereal – Network Protocol Analyzer* (*Analizador de Protocolos de Rede*), poderosa aplicação que faz análise do tráfego de dados da rede e reconhece centenas

de protocolos. Esta ferramenta está disponível para as plataformas *Windows, Linux, BSD* e *Mac*.

6.3 – ANÁLISE DOS PACOTES TCP/RMI COLETADOS NA REDE

Com a ajuda do software de análise de tráfego (*Ethereal*), chegaram-se aos gráficos das figuras 6.5, 6.6 e 6.7.

Na figura 6.5, é apresentada às informações coletadas pelo software *Ethereal*, dos pacotes TCP/RMI na comunicação cliente/servidor. O tempo de coleta das informações foi de 123,281 segundos (00:02:03). Neste intervalo, foram coletados 2133 pacotes TCP/RMI perfazendo um total de 188277 bytes (183,63 Kbytes), tendo como tamanho médio dos pacotes transmitidos de 1,49 Kbytes/s.

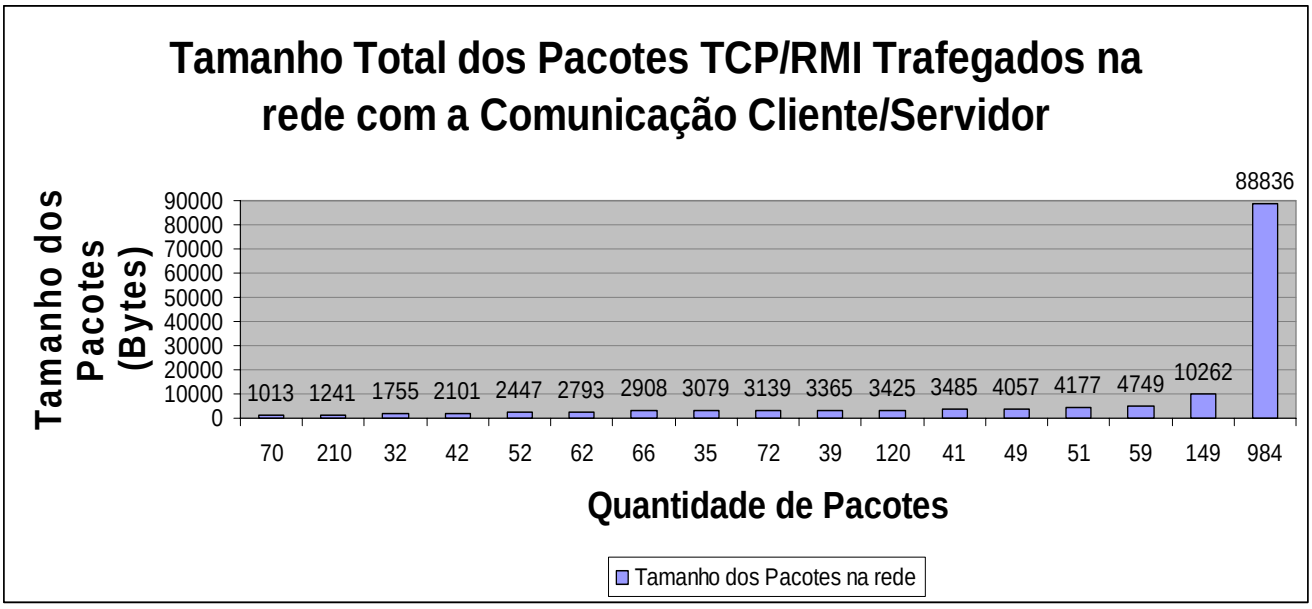


Figura 6.5: Tamanho dos pacotes TCP/RMI na comunicação cliente/servidor.

Na figura 6.6, apresenta-se a demanda de pacotes enviados pelo servidor à máquina cliente, e a quantidade de quadros enviados pelo cliente ao servidor solicitando a execução de alguma tarefa. Estas trocas de mensagens são oriundas do protocolo TCP, que é um serviço confiável. Com isso, há a necessidade na troca de informações, que o destino envie uma

confirmação positiva ao destinatário para que a transmissão de novos pacotes se torne possível. Em caso de confirmação negativa o pacote é reenviado. Na figura 7.6, “A” representa o cliente e “B” o servidor. Nota-se que foram enviados 1158 pacotes ao servidor e este enviou um total de 975 pacotes ao cliente.

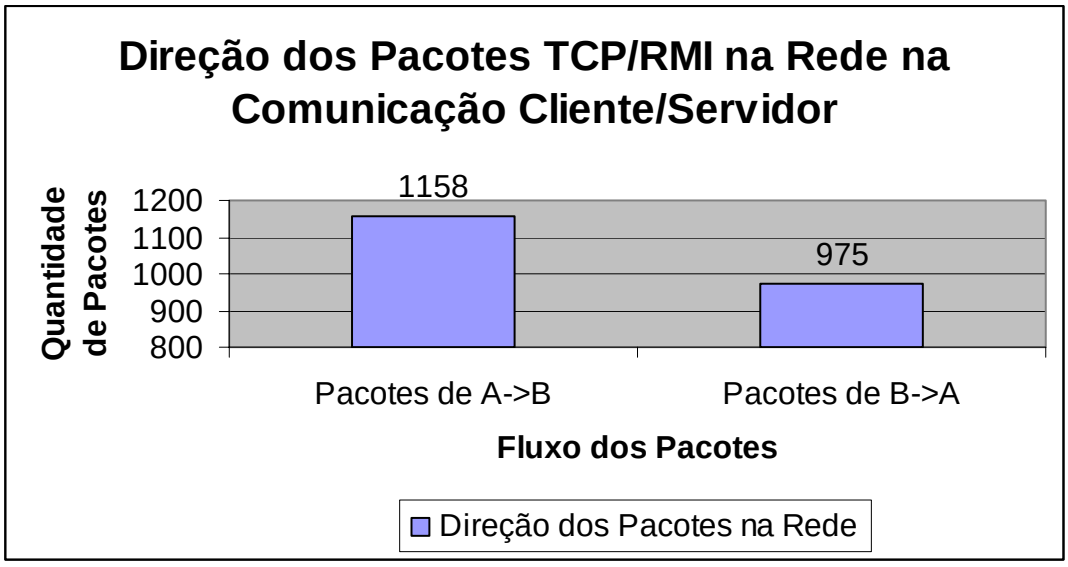


Figura 6.6: Direção dos Pacotes TCP/RMI

Na figura 6.7 é ilustrado o tamanho total em *bytes* dos pacotes enviados do cliente “A” ao servidor “B”, e retornando estas informações do servidor “B” ao cliente “A”.

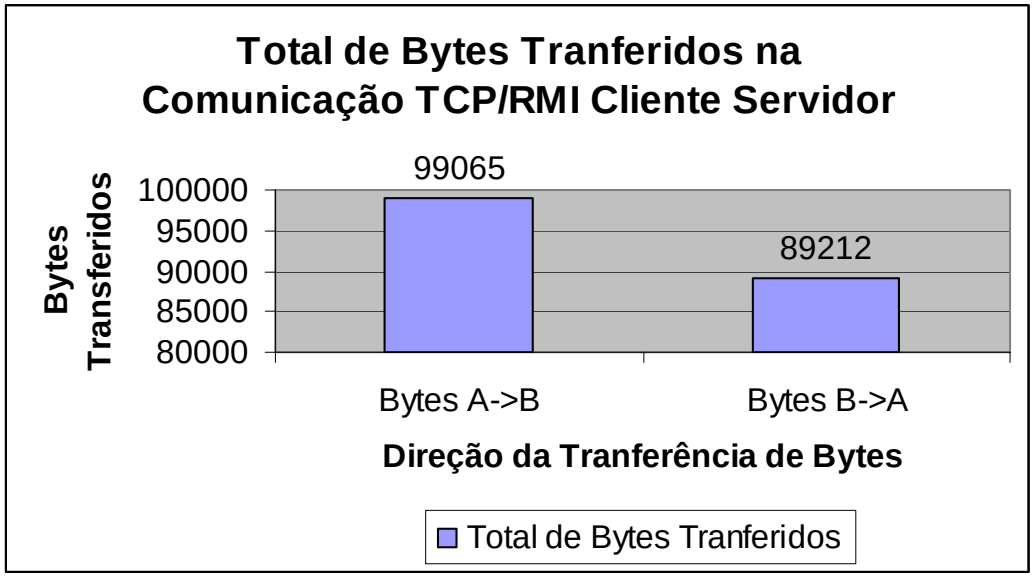


Figura 6.7: Total de *bytes* transferidos na comunicação TCP/RMI.

6.4 – ANÁLISE DOS PACOTES UDP/RTP COLETADOS NA REDE

A transmissão dos pacotes enviados pelo servidor à rede de computadores utilizando o protocolo UDP/RTP, é um serviço não confiável, pois, não garante a entrega dos pacotes enviados ao destino especificado. Os dados coletados na comunicação entre cliente/servidor na figura 6.8, apresenta um total de 13679 pacotes enviados, acumularam um total de 13042168 bytes transmitidos ou 12,42 Mbytes. A média de *bytes* transmitidos foi de 103,54 Kbytes/s.

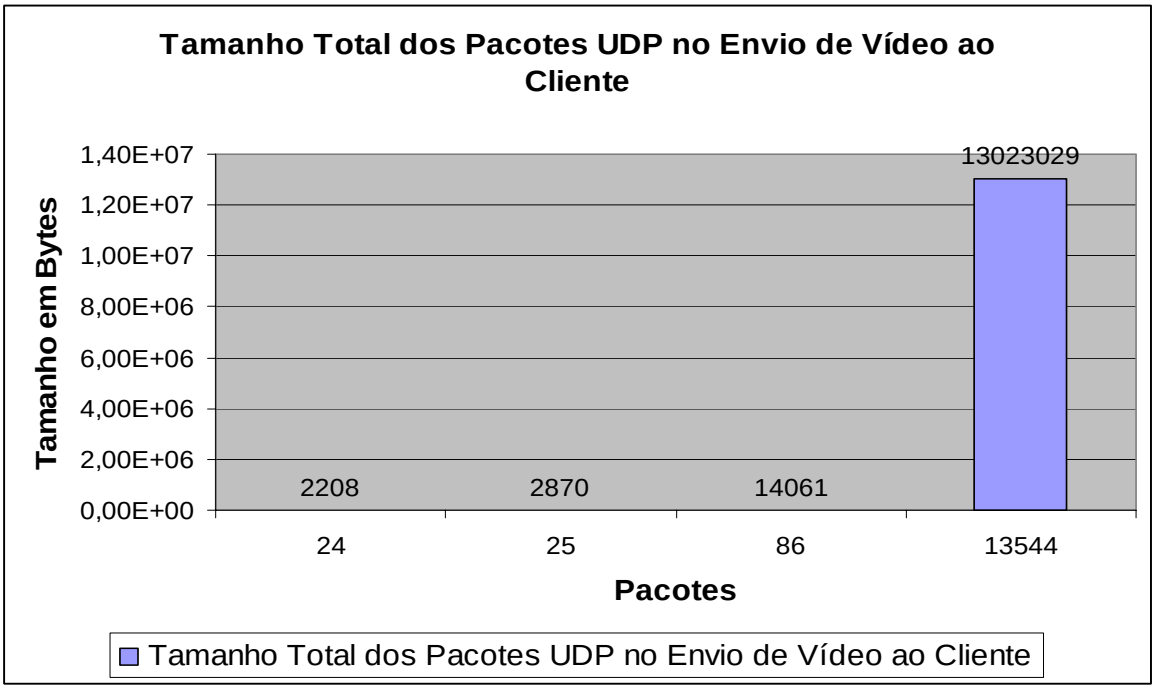


Figura 6.8: Tamanho dos Pacotes UDP.

6.5 – ANÁLISE DO FLUXO DE PACOTES DOS PROTOCOLOS DE COMUNICAÇÃO

A figura 6.9 apresenta o fluxo de dados obtidos na rede. Pode-se salientar que, a quantidade de dados do protocolo UDP/RTP foi mais expressiva por ser pacotes de conteúdo multimídia, e assim sendo, requer um canal de comunicação orientado a conexão. Já o

protocolo TCP/RMI utiliza um canal de comunicação não orientado à conexão para a troca de mensagens. O protocolo TCP/RMI precisa ser invocado para que haja à troca de mensagens.

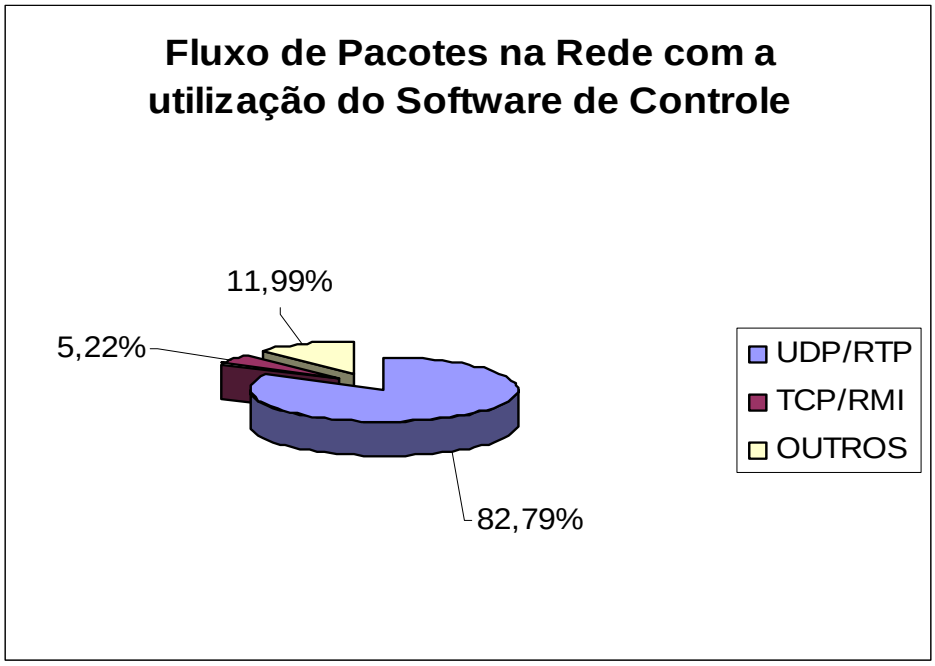


Figura 6.9: Fluxo de Pacotes na rede.

Na figura 6.10 apresenta-se o fluxo global de pacotes em *bytes* na rede de computadores.

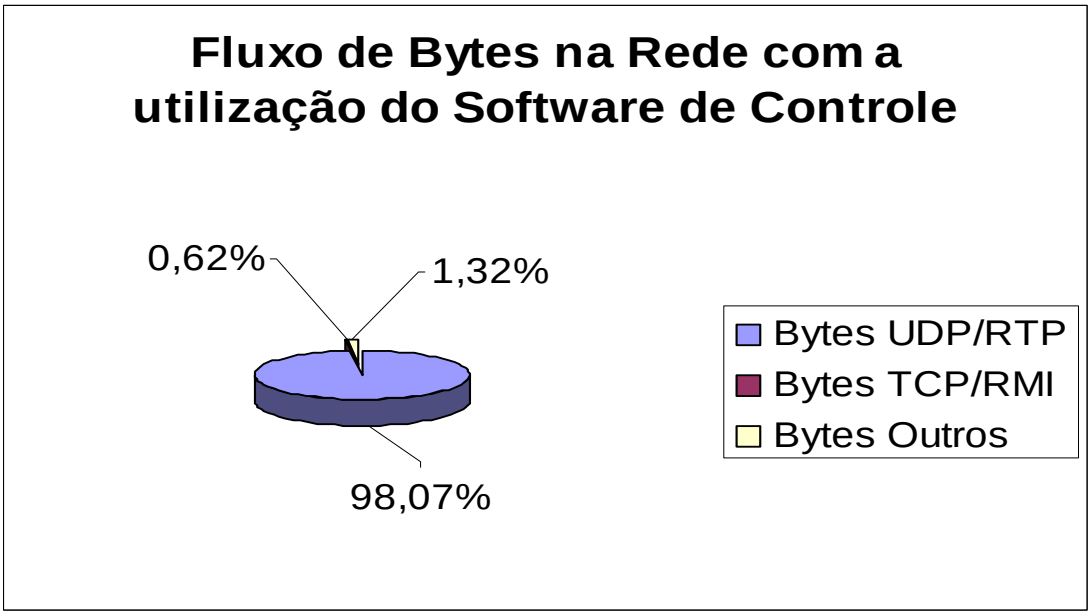


Figura 6.10: Fluxo de bytes na rede.

6.6 – CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresentou as informações coletadas do fluxo de dados entre cliente/servidor e o tempo de acionamento dos motores utilizando o sistema de controle do SCORBOT ER – III. Pode-se observar que o tempo para a execução de uma tarefa em uma LAN se manteve desprezível, não causando danos na operação solicitada. Já o fluxo de pacotes na rede foi satisfatório, mostrando que para uma conexão discada a transferência de dados se tornaria insatisfatória. Assim, para o funcionamento do software é necessário um canal com uma transferência significativa para que não haja a perda de informações (vídeo) e nem atraso nas aplicações (tarefas).

CAPÍTULO 7 - CONSIDERAÇÕES FINAIS

Os trabalhos e estudos realizados, que resultaram na criação do software de controle para a manipulação do braço mecânico SCORBOT ER – III, envolveram o desenvolvimento de um conjunto de programas (classes JAVA e uma biblioteca dinâmica – dll) e um novo circuito de controle que, integrados, propiciam ao usuário controlar remotamente o dispositivo robótico através da porta paralela do computador. Nesta etapa, o circuito de controle seleciona qual dos motores será ativado para movimentação, a partir da solicitação do software cliente ao servidor.

Na criação deste software para a manipulação do braço mecânico, utilizaram-se técnicas que buscaram ao máximo aumentar o grau de portabilidade do aplicativo. Entretanto, tal objetivo não foi possível devido a não utilização do pacote JAVAX.COMM, o qual, possui erros de implementação na sua versão atual. Assim, utilizou-se uma biblioteca dinâmica – DLL, feita com comandos ANSI da linguagem C++, para que o aplicativo pudesse enviar os *bits* necessários à porta paralela do computador para a ativação de um motor específico, através do circuito de controle. A ligação desta biblioteca com a linguagem JAVA se deu através de métodos nativos – JNI. Já a comunicação entre cliente/servidor foi implementada utilizando conceitos de métodos remotos – RMI. Para a execução de tarefas pré-definidas pelo robô, utilizou-se conceitos de JDBC para o armazenamento de instruções que visam automatizar trajetórias de motores específicos. Por fim, monitorou-se os movimentos do braço remotamente através de uma WebCam Digital, utilizando o Protocolo de Tempo Real – RTP da biblioteca JMF.

Análises do software indicaram que a latência na comunicação para ativação dos motores do robô se manteve desprezível em uma rede local. No mesmo tipo de rede a transferência de quadros de imagens utilizando o protocolo RTP foi de boa qualidade,

proporcionando ao cliente uma nítida visualização dos movimentos do robô remotamente. Já em uma conexão discada, a transferência de imagens se torna não confiável, pois, ocorre a perda de muitos pacotes na rede.

Os temas relacionados à comunicação remota entre dispositivos eletrônicos devem ser vistos como um lugar em destaque, não apenas para pessoas que trabalham na área de automação, controle e informática, mas também para aquelas que, de alguma forma, utilizam a robótica como ferramenta para facilitar seus trabalhos.

Para trabalhos futuros propõe-se:

- ✓ A leitura dos *encoders* através da porta paralela do computador;
- ✓ A criação de uma interface de cadastramento de usuário através da internet;
- ✓ Implementação *multicast* para o envio de imagens para um grupo de computadores pertencentes a rede;
- ✓ Estudo e implementação de métodos para a visualização do robô em profundidade através de espelhos;
- ✓ Estudo e implementação da biblioteca JAVAX.COMM para a leitura dos *bits* na porta paralela;
- ✓ Programação para desvios na detecção do sistema operacional corrente, para o contorno do problema de portabilidade entre diferentes plataformas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] – K. Goldberg, K. Mascha, M. Genter, N. Rothenberg, C. Sutter, and J. Wiegley. Desktop teleoperation via the World Wide Web. In *Proceedings of IEEE International Conference on Robotics and Automatio*n, May 1995a.

[2] – B. Dalton, *Techniques for WEB Telerobotics*, Tese de Doutorado, Department of Mechanical and Materials Engineering University of Western Australia, Perth, Australia, 2001.

[3] – Sun Microsystems. *The JAVA Tutorial.* Disponível na Internet via WWW. URL: http://JAVA.sun.com/docs/books/tutorial/index.html.

[4] – H. Hu, L. Yu, P. W. Tsui, and Q. Zhou, Internet-based Robotic System for Teleoperation *In International Journal of Assembly Automation, Vol. 21, No. 2,* pages 143-152, 2001.

[5] – A. Malinowski and B. Wilamowski, Controlling Robots via Intenet, In 1 st International Conference on Information Technology in Mechatronics, Istanbul, Turkey, October 1-3, pages 101-107, 2001.

[6] – P. G. Backes, K. S. Tso and G. K. Tharp, The WEB Interface for Telescience, In *Presence, Vol. 8, No. 5, pages 531-539, October, 1999.*

[7] – GUARDIA, L. E., *Controle de um Braço Mecânico SCORBOT III utilizando um computador PC*, UNESP, Setembro, 2002.

[8] – http://super.abril.com.br/especiais/1197/futuro25.html. Acesso em 10/12/2005.

[9] – http://www.din.uem.br/ia/vida/robotica2/classif.htm. Acesso em 11/12/2005.

[10] – G. CORNELL, C. S. HORSTMANN, *Core JAVA – Vol I - Fundamentos,* Editora – Makron Books, São Paulo, 2003.

[11] – GREMMELMAIER, C. H., *Introdução à Programação em Linguagem JAVA,* Erechin, 2002.

[12] – Nascimento, F. A; Rodrigues, M., LINGUAGEM DE PROGRAMAÇÃO JAVA – UNIVERSIDADE LUTERANA DO BRASIL, Canoas – RS, 2003

[13] – CHELLA, M. T., A*mbiente de Robótica para Aplicações Educacionais com SuperLogo,* UNICAMP, Campinas, SP, 2002.

[14] – CORNELL, C. S. HORSTMANN, *Core JAVA – Vol II - Recursos Avançados,* Editora – Makron Books, São Paulo, 2003.

[15] – J. D. Ringgenberg, *Object-Oriented Design of Portable Control System Software.* Tese de Mestrado, Mechanical Engineering, University of California – Berkeley, Spring 2001.

[16] – DEL CID PORTILLO, J. *Parallel printer port access through*. Disponível em <http://www.geocities.com/juanga69/parport/>.Acesso em 2005.

[17] – MECATRÔNICA FÁCIL. Utilizando o logo em windows 2000, NT e XP. Disponível em: <http://www.mecatronicafacil.com.br/downloads/logo_instr.htm>.Acesso em 2005.

[18] – JAVA *Media FrameWork API guide* – Sun Microsystems, 1999.

[19] – Booch, G.; Rumbauch, J.; Jacobson, I., *The Unified Modeling Language User Guide,* Addison Wesley, 1998.

[20] – http://pt.wikipedia.org/wiki/Diagrama_de_objetos. Acesso em 05/01/2006.