



UNIVERSIDADE ESTADUAL PAULISTA

Programa de Pós-Graduação em Engenharia Elétrica

**Emprego da Tecnologia Java para Implementar a Parte
Lógica de um Processador de Aplicação com Capacidade de
Operar em Rede de Comunicação (NCAP), em
Conformidade com o Padrão IEEE 1451.**

Edson Antonio Batista

Ilha Solteira - SP

1210001519





UNIVERSIDADE ESTADUAL PAULISTA

"JÚLIO DE MESQUITA FILHO"
FACULDADE DE ENGENHARIA
DE ILHA SOLTEIRA



DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEE

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**Emprego da Tecnologia Java para Implementar a Parte
Lógica de um Processador de Aplicação com Capacidade de
Operar em Rede de Comunicação (NCAP), em
Conformidade com o Padrão IEEE 1451.**

Dissertação submetida à Faculdade de Engenharia de Ilha Solteira da Universidade Estadual Paulista -UNESP, como parte dos requisitos necessários para obtenção do título de Mestre em Engenharia Elétrica.

Candidato: Eng. Eletr. Edson Antonio Batista

Orientador: Eng. Eletr. Alexandre César Rodrigues da Silva – FEIS / UNESP

Co-orientador: Prof. Dr. Cláudio Kitano – FEIS / UNESP

Ilha Solteira, Maio de 2004.

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação/Serviço Técnico de Biblioteca e Documentação da UNESP-Ilha Solteira

Batista, Edson Antonio

B333e Emprego da tecnologia Java para implementar a parte lógica de um processador de aplicação com capacidade de operar em rede de comunicação (NCAP), em conformidade com o padrão IEEE 1451 / Edson Antonio Batista. -- Ilha Solteira : [s.n.], 2004
xiii, 127 p. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2004.

Orientador: Alexandre César Rodrigues da Silva

Co-orientador: Cláudio Kitano

Bibliografia: p. 111-115

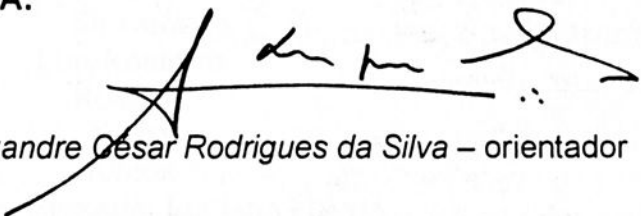
1. Java (Linguagem de programação de computador). 2. Padrão IEEE 1451. 3. NCAP.

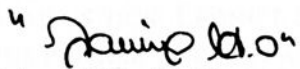
***Emprego da Tecnologia Java para Implementar a
Parte Lógica de um Processador de Aplicação com
Capacidade de Operar em Rede de Comunicação
(NCAP), em Conformidade com o Padrão IEEE
1451***

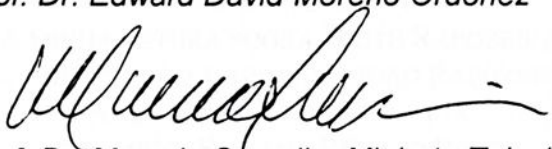
Edson Antonio Batista

**DISSERTAÇÃO SUBMETIDA À FACULDADE DE ENGENHARIA - CAMPUS DE
ILHA SOLTEIRA – UNESP – COMO PARTE DOS REQUISITOS NECESSÁRIOS
PARA A OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA ELÉTRICA**

COMISSÃO EXAMINADORA:


Prof. Dr. Alexandre Cesar Rodrigues da Silva – orientador


Prof. Dr. Edward David Moreno Ordonez


Prof. Dr. Marcelo Carvalho Minhoto Teixeira

Ilha Solteira-SP, maio de 2004.



1210001519



Proc. 063/04 - NPD 151/04

UNESP - "CAMPUS DE ILHA SOLTEIRA"	
SERVIÇO TÉC. DE BIBLIOTECA E DOCUMENTAÇÃO	
DATA DE CHEGADA	DATA DE TOMBO
<i>14.09.04</i>	<i>30.09.04</i>
REGISTRADO POR	TOMBO
<i>ailza</i>	<i>Te. 1519</i>
AQUISIÇÃO	CLASSIFICAÇÃO
<i>hloação</i> <i>autor</i> <i>R\$10.00</i>	<i>B333e</i>

Co sys 226514
sys 56568
30405025

AOS MEUS PAIS: EUNICE E OVÍDIO
 AOS MEUS IRMÃOS (POR ORDEM DE CHEGADA):

CRISTINA
 ANA LÚCIA
 SILVANA
 LUIS ROBERTO
 ROSIANE
 NÚBIA
 JUNIOR

A MINHA NAMORADA: LUCIANA FLÁVIA
 AOS MEUS TIOS: LEONICE E JOÃO
 AOS MEUS PRIMOS: LUIS EDUARDO E RITA
 AO MEU CUNHADO: EDSON VINICIUS
 A MINHA FUTURA SOGRA: RUTH RAPOZEIRA
 AO MEU FUTURO SOGRO JOSÉ JOÃO RAPOZEIRA
 A MINHA CUNHADA: CRISTINA
 AO AMIGO: SILVANO RENATO ROSSI
 A TODOS OS MEUS AMIGOS

DEDICO



Agradecimento

Agradeço a DEUS pela saúde, pela oportunidade, pela força e por ter posicionado em meu caminho as amizades que contribuíram para a realização deste trabalho.

Agradeço aos meus pais, Ovídio e Eunice, pelo amparo, pela proteção e confiança que fizeram e estão fazendo para que eu consiga alcançar mais etapas.

Aos meus irmãos, Cristina, Ana, Silvana, Roberto, Rosiane, Núbia, Junior, pelo incentivo, apoio familiar e pela nossa união. Muito obrigado.

A minha namorada, Luciana Flávia, pela atenção, por existir e ser a inspiração da minha vida, beijos...

À minha futura sogra Ruth e meu futuro sogro João, pela amizade e pelo incentivo.

Ao meu futuro cunhado Edson Vinícius, pela amizade e pela convivência.

Ao Engenheiro Eletricista Alexandre César Rodrigues da Silva, pela incansável orientação, pelos esclarecimentos, pela confiança, pela amizade, pelos incentivos e por ser um exemplo de profissional.

Ao Prof. Dr. Cláudio Kitano pela co-orientação, pelo interesse e pela atenção dedicada a este trabalho.

Ao Prof. Dr. Aparecido Augusto de Carvalho pelos incentivos, por possibilitar a realização deste trabalho, pelo apoio técnico e por ser um modelo de profissional.

Ao Engenheiro Eletromecânico, Silvano Renato Rossi, pelo apoio técnico, pelo incentivo e por contribuir diretamente para minha qualificação profissional.

Aos amigos de laboratório, Carlos Eduardo, João Marcos, André, Adriano, pela saudável convivência profissional.

À Universidade Estadual Paulista e ao Departamento de Engenharia Elétrica pela oportunidade de realizar o curso.

Ao CNPQ, pelo apoio financeiro cedido através de Bolsa de Estudo durante o curso.

Ao Núcleo de Automação e Controle da Uniderp-MS, pelo apoio técnico...

À Fundunesp, pelo apoio financeiro cedido para compra de equipamentos para desenvolver este trabalho.

Espero poder retribuir a todos a felicidade e a sensação de dever cumprido que sinto neste momento e da qual jamais me esquecerei.



Resumo

Este trabalho apresenta a implementação de um processador de aplicação com capacidade de operar em rede de comunicação, no padrão IEEE 1451 denominado de NCAP. O NCAP é um nó de rede que possibilita o controle das atividades de um transdutor via Internet. Para desempenhar suas funções, o NCAP é construído com o emprego de sistemas abertos e ferramentas padronizadas. O padrão IEEE1451 visa resolver a incompatibilidade existente entre diferentes fabricantes da área da Instrumentação Distribuída Inteligente.

Segundo o modelo de informação IEEE 1451.1, o NCAP compõe-se das partes lógicas e físicas. O objetivo principal do projeto foi utilizar a linguagem de programação Java no desenvolvimento de um software, para implementar a parte lógica do NCAP. Neste projeto utilizou-se um microcomputador e um dispositivo FPGA para construir a parte física. A parte lógica do processador, aqui denominada de software NCAP, controla tanto o acesso à rede de comunicação, como as atividades de um dispositivo FPGA. Este dispositivo, denominado no projeto de Gerenciador de Protocolo, tem a finalidade de proporcionar um drive para a Interface Independente do Transdutor (TII). Na TII é conectado o módulo de interface para transdutor inteligente (STIM). Neste módulo, conecta-se transdutor. O uso da linguagem Java, para desenvolver todo o software NCAP, alcançou resultados importantes para se construir um NCAP de baixo custo e de fácil manutenção. Outra tecnologia importante no projeto é a rede Ethernet preconizada como uma tendência para as redes de comunicação industrial, devido à evolução, a grande popularidade e ao baixo custo. No projeto, através da Ethernet, o NCAP deve possibilitar aos usuários conectados na Internet o controle das atividades de um transdutor inteligente.

Palavras-chave: Tecnologia Java, Padrão IEEE1451, NCAP, Transdutor Inteligente, Ethernet, FPGA.



Abstract

This work presents an implementation of a Network Capable Application Processor (NCAP) according to the IEEE1451 standard. The NCAP is a network node that allows to control transducer tasks through the Internet. In order to develop its functions, the NCAP must be built on open systems and standardized tools. The IEEE1451 standard aims to solve the existing incompatibility among different manufacturers in the Smart Distributed Instrumentation area. According to the inquiry model IEEE1451.1, the NCAP is constituted by a logic part and a physical part. Using the Java language program to develop a software in order to improve the NCAP logic part, is the main purpose of this project. In this work a Personal Computer and FPGA device for building the physical part of NCAP is used. The logic part of the processor, here denominated NCAP software controls the network communications, as well as the FPGA device. This device denominated Protocol Manager has the purpose of implement the communication between the NCAP module and the smart transducer. The transducer is connected to this module. The use of Java language to develop the entire NCAP software, attained important result to manufacture a NCAP of low cost and an easy maintenance. Other important technology at this project is Ethernet commended to a trend to network of industrial communication, due to evolution, great popularity and low cost. Through Ethernet network the NCAP must allow that clients connected to the Internet to control the tasks of a smart transducer.

Keywords: Java Technology, IEEE1451 Standard, NCAP, Smart Transducer, Ethernet, FPGA



Lista de Figuras

Figura 1.1: Configuração do padrão IEEE1451.----- 2

Figura 2.1: O modelo de Referência RM-OSI. ----- 8

Figura 2.2: Topologia em anel. -----12

Figura 2.3: Topologia em Barramento. -----13

Figura 2.4: Topologia Estrela. -----14

Figura 3.1: Modelo de arquitetura IEEE 802.-----15

Figura 3.2: Rascunho da ethernet. -----16

Figura 3.3: Estados de um quadro Ethernet. -----18

Figura 3.4: Funcionamento do CSMA/CD. -----19

Figura 3.5: Padrão de formação de um quadro de bits (802.3).-----20

Figura 3.6: Detecção de colisão.-----21

Figura 3.7: Cabo coaxial grosso. -----25

Figura 3.8: Tecnologia 10Base5. -----26

Figura 3.9: Configuração para o 10Base2. -----26

Figura 3.10: Tecnologia 10Base T.-----27

Figura 3.11: Foto de uma fibra óptica.-----28

Figura 3.12: Arquitetura de comunicação cliente/servidor. -----32

Figura 3.13: Configuração de um sistemas distribuído. -----33

Figura 4.1: A arquitetura da tecnologia Java. -----38

Figura 4.2: Utilizando o comando *import* em Java. -----45

Figura 4.3: Exemplo de programação Java. -----53

Figura 4.4: Etapas para compilar e executar o programa “*Mensagem*”. -----54

Figura 4.5: Execução do programa “*Mensagem*”. -----54

Figura 4.6: Descrição do programa “*Contagem*”.-----55

Figura 4.7: Etapas necessárias para compilar e executar o programa “*Contagem*”.-----55

Figura 4.8: Resultado da execução do programa “*Contagem*”.-----56

Figura 4.9: Arquitetura RMI.-----60

Figura 4.10: Código fonte do programa *InterfaceEnv*.-----61



Figura 4.11: Código fonte do programa <i>ServidorEnv</i> . -----	62
Figura 4.12: Código fonte do programa <i>ClienteEnv</i> .-----	63
Figura 4.13: Etapas de compilação e registro do <i>ServidorEnv</i> . -----	63
Figura 4.14: Etapa de execução do programa <i>ServidorEnv</i> . -----	64
Figura 4.15: Etapas para compilar e executar o programa <i>ClienteEnv</i> .-----	64
Figura 4.16: O servidor recebe a mensagem de um cliente. -----	65
Figura 4.17: Código fonte de um programa que utiliza o pacote <i>javacomm</i> . -----	67
Figura 4.18: Etapas para compilar e executar o programa <i>ListaPorta</i> . -----	67
Figura 4.19: Resultado da execução do programa <i>ListaPorta</i> em um determinado Pc. -----	68
Figura 5.1: Maneira como a família de padrões IEEE 1451 estão relacionados. -----	70
Figura 5.2: Primeira configuração de um NCAP. -----	78
Figura 5.3: Configuração de um NCAP, em conformidade com o padrão IEEE1451.1. -----	80
Figura 6.1: Conector DB-25. -----	83
Figura 6.2: Configuração do projeto controlado pelo software NCAP. -----	86
Figura 6.3: Diagrama de classe para desenvolver o software NCAP.-----	87
Figura 6.4: Código fonte da interface remota <i>IEEE1451_BaseTransducerBlock</i> . -----	88
Figura 6.5: Classes utilizadas no programa <i>IEEE1451_NCAPBlock</i> .-----	89
Figura 6.6: Herança e implementação do programa <i>IEEE1451_NCAPBlock</i> .-----	89
Figura 6.7: Etapa de identificação dos clientes. -----	90
Figura 6.8: Seleciona-se a porta LPT1 para realizar a comunicação. -----	91
Figura 6.9: O endereço de função é enviado para LPT1. -----	92
Figura 6.10: Construção do objeto remoto.-----	93
Figura 6.11: Classes para compilar o programa <i>IEEE1451_FunctionBlock</i> . -----	94
Figura 6.12: Método <i>enviaIp()</i> captura endereço IP da máquina.-----	94
Figura 6.13: Interface para o cliente. -----	95
Figura 6.14: Etapa para localizar o servidor. -----	96
Figura 6.15: Implementação do objeto remoto “ <i>adquiri</i> ”. -----	96
Figura 6.16: Tratamento de Exceção para conexão remota. -----	97
Figura 6.17: Etapas para compilar e registrar o <i>IEEE1451_NCAPBlock</i> . -----	97
Figura 6.18: Mostra a execução do programa <i>IEEE1451_NCAPBlock</i> . -----	97
Figura 6.19: Etapas para compilar e executar o <i>IEEE1451_FunctionBlock</i> .-----	98

Figura 6.20: Servidor identifica um cliente.-----98

Figura 6.21: Execução do programa *IEEE1451_FunctionBlock*.-----99

Figura 6.22: O cliente envia uma função ao gerenciador de protocolo. ----- 100

Figura 6.23: O servidor transfere o valor “160” ao dispositivo. ----- 101

Figura 6.24: Simulação da lógica de endereçamento. ----- 102

Figura 6.25: Execução do *AcionaCooler*. ----- 103

Figura 6.26: Arquitetura para testar o software. ----- 103

Figura 6.27: Arquitetura conformidade com o padrão IEEE1451. ----- 104



Lista de Tabelas e Siglas

Tabela 3.1: Tentativa de transmissão na rede Ethernet.	23
Tabela 3.2: Cabeamento para 10Mbps no padrão 802.3.	24
Tabela 3.3: Cabeamento para a rede Fast Ethernet.	29
Tabela 3.4: Cabeamento para o Gigabit Ethernet.	30
Tabela 4.1: Pacotes pertencentes à linguagem de programação Java.	44
Tabela 4.2: Palavras-chaves.	45
Tabela 4.3: Tipos de dados primitivos em Java.	46
Tabela 4.4: Operadores Aritméticos.	46
Tabela 4.5: Operadores Lógicos.	47
Tabela 4.6: Operadores de Comparação.	47
Tabela 4.7: Operadores de Atribuição.	48
Tabela 4.8: Seqüência de escape.	48
Tabela 4.9: Operadores de incremento e decremento.	48
Tabela 5.1: Evolução do padrão IEEE1451.	72
Tabela 5.2: Pinagem recomendada para a TII.	76
Tabela 5.3: Descrição da pinagem da TII.	77
Tabela 6.1: Pinagem do conector DB-25.	84



Lista de Abreviaturas e Siglas

A/D: Analógica/Digital

API: Applications Programming Interface (Interface de Programas Aplicativos)

ASIC: Application Specific Integrated Circuit (Circuito Integrado de Aplicação Específica)

CAN: Controller Area Network

CSMA/CD: Carrier Sense Multiple Access with Collision Detection (Múltiplo Acesso com Verificação de Portadora com Detecção de Colisão)

D/A: Digital/Analógica

DMC: Distributed Measurement and Control (Medição e Controle Distribuído)

FPGA: Field Programmable Gate Array (Matriz de Portas Lógicas Programáveis em Campo)

GUI: Graphical User Interface (Interface Gráfica com o Usuário)

IDL: Interface Definition Language (Linguagem de Definição de Interfaces)

I/O: Input/Output (Entradas / Saídas)

IEEE: Institute of Electrical and Electronics Engineers (Instituto de Engenheiros Eletricistas e Eletrônicos)

IP: Internet Protocol (Endereço na Internet)

ISO: International Standards Organization (Organização Internacional de Padronização)

JDK: Java Development Kit (Kit de desenvolvimento Java)

JVM: Java Virtual Machine (Máquina Virtual Java)

JRE: Java Run-Time Environment (Ambiente para execução Java)

LAN: Local Area Network (Redes Locais)

LLC: Logic Link Control (Controle de Enlace Lógico)

MAC: Medium Access Control (Controle de Acesso ao Meio)

MPU: Microprocessor Unit (Unidade Microprocessadora)

NCAP: Network Capable Application Processor (Processador de Aplicação com Capacidade de Operar em Rede)

NIST: National Institute of Standards and Technology (Instituto Nacional de Padrões e Tecnologias)



OO: Object Oriented (Orientado a Objeto)

PC: Personal Computer (Computador Pessoal)

PLD: Programmable Logic Device (Dispositivo de Lógica Programável)

RMI: Remote Method Invocation (Invocação de Métodos Remotos)

RM-OSI: Reference Model-Open Systems Interconnection (Modelo de Referência-Sistema Aberto de Interconexão)

STIM: Smart Transducer Interface Module (Módulo de Interface para Transdutores Inteligentes)

TCP/IP: Transmission Control Protocol/Internet Protocol (Protocolo de Controle de Transmissão de Protocolo/Endereço na Internet)

TEDS: Transducer Electronic Data Sheet (Especificações de Transdutor em Formato Eletrônico)

TII: Transducer Independent Interface (Interface Independente para Transdutores)

VHDL: Very High Speed Integrated Circuit - Hardware Description Language
(Linguagem de descrição de Hardware para circuitos integrados de velocidade muito elevada *VHSIC*)



Sumário

CAPÍTULO - 1.....	1
1. INTRODUÇÃO	1
CAPÍTULO - 2.....	7
2. REDE DE COMPUTADORES.....	7
2.1 INTRODUÇÃO	7
2.2 CAMADAS PERTENCENTES AO RM-OSI	9
2.2.1 A Camada Física	9
2.2.2 A Camada de Enlace de Dados	9
2.2.3 A Camada de Rede.....	9
2.2.4 A Camada de Transporte.....	10
2.2.5 A Camada de Sessão.....	10
2.2.6 A Camada de Apresentação.....	10
2.2.7 A Camada de Aplicação	11
2.3 TOPOLOGIA DE UMA REDE DE COMUNICAÇÃO.....	11
2.3.1 Topologia em Anel.....	12
2.3.2 Topologia em Barramento.....	13
2.3.3 Topologia Estrela	13
CAPÍTULO - 3.....	15
3. PADRONIZAÇÃO DAS REDES LOCAIS	15
3.1 INTRODUÇÃO	15
3.2 O PADRÃO IEEE 802.3	16
3.3 ESPECIFICAÇÕES DA REDE ETHERNET	17
3.3.1 Modo de Transmissão.....	17
3.3.2 Controle de Acesso ao Meio.....	18
a) A subcamada MAC 802.3	18
b) Algoritmo CSMA/CD	19



c) <u>Padronização do Quadro de Bits</u>	20
3.3.3 <u>Meio Físico</u>	24
a) <u>Ethernet 10 Mbps</u>	24
b) <u>Fast Ethernet</u>	28
c) <u>Gigabit Ethernet</u>	29
3.4 <u>TIPOS DE COMUNICAÇÃO PROPOSTOS NO PADRÃO IEEE1451</u>	31
3.4.1 <u>Cliente/Servidor</u>	31
3.4.2 <u>Sistemas Distribuídos SD</u>	33
4. <u>TECNOLOGIA JAVA</u>	35
4.1 <u>INTRODUÇÃO AO JAVA</u>	35
4.2 <u>PECULIARIDADES DA LINGUAGEM DE PROGRAMAÇÃO JAVA</u>	37
4.2.1 <u>A JVM</u>	37
4.2.2 <u>Coletor de Lixo</u>	38
4.3 <u>CARACTERÍSTICAS DA LINGUAGEM DE PROGRAMAÇÃO JAVA</u>	38
4.3.1 <u>Orientada a Objeto</u>	39
a) <u>Classe</u>	39
b) <u>Encapsulamento</u>	40
c) <u>Polimorfismo</u>	40
d) <u>Hereditariedade</u>	40
4.3.2 <u>Distribuída</u>	41
4.3.3 <u>Segura</u>	41
4.3.4 <u>Robusta</u>	42
4.4 <u>INSTALAÇÃO DE UM KIT DE DESENVOLVIMENTO JAVA</u>	43
4.4.1 <u>Pacotes da linguagem Java</u>	44
4.5 <u>SINTAXES DA PROGRAMAÇÃO JAVA</u>	45
4.6 <u>ESTRUTURAS DE SELEÇÃO EM JAVA</u>	49
4.6.1 <u>Estrutura de seleção if</u>	49
4.6.2 <u>Estrutura de seleção if/else</u>	50
4.6.3 <u>Estrutura de repetição while</u>	50
4.6.4 <u>Estrutura de repetição for</u>	51
4.6.5 <u>Estrutura seleção múltipla switch</u>	51

4.6.6 Estrutura de repetição do/while.....	52
4.7 EXEMPLO DE PROGRAMAÇÃO JAVA.....	52
4.8 JAVA RMI	56
4.8.1 Introdução.....	56
4.8.2 Objetivos Java RMI.....	57
4.8.3 Vantagens e desvantagens da Java RMI.....	57
4.8.4 Camada Skeleton e Stub.....	58
4.8.5 Camada de referência remota.....	59
4.8.6 Camada de Transporte	59
4.8.7 Exemplo de uma aplicação RMI.....	60
4.8.8 API de Comunicação Java.....	65
<u>CAPÍTULO - 5.....</u>	69
<u>5. PADRÃO IEEE1451</u>	69
5.1 INTRODUÇÃO	69
5.2 OBJETIVO DO PADRÃO IEEE 1451	69
5.3 HISTÓRIA DO PADRÃO IEEE1451.....	70
5.4 EVOLUÇÃO DO PADRÃO IEEE 1451.....	71
5.5 O PADRÃO IEEE1451.2	73
5.5.1 STIM - Módulo Interface para Transdutores Inteligentes.....	73
a) Microcontrolador.....	73
b) TEDS.....	74
5.5.2 TII – Interface Independente do Transdutor.....	75
5.6 O PADRÃO IEEE 1451.1 - NCAP	77
<u>CAPÍTULO - 6.....</u>	82
<u>6. RESULTADOS.....</u>	82
6.1 IMPLEMENTAÇÕES	82
6.2 SOFTWARE NCAP.....	86
6.3 TRABALHOS FUTUROS	105
6.3.1 Porta USB	105



<u>6.3.2 Jini.....</u>	106
<u>6.3.3 CORBA</u>	106
<u>6.3.4 Processador Java.....</u>	106
<u>6.3.5 Interface gráfica</u>	107
<u>6.3.6 Applet</u>	107
<u>6.3.7 Cliente via celular.....</u>	107
<u>6.3.8 NCAP Java x C++</u>	107
<u>6.3.9 Can</u>	107
<u>6.3.10 Java 3D e Java Advance</u>	108
<u>6.3.11 Banco de Dados.....</u>	108
<u>6.3.12 Desenvolver API Java</u>	108
<u>CAPÍTULO - 7.....</u>	109
<u>7. CONCLUSÃO</u>	109
<u>REFERÊNCIAS.....</u>	111
<u>APÊNDICE A - SITES PARA ASSUNTO JAVA</u>	116
<u>APÊNDICE B – PROGRAMAS EM JAVA.....</u>	117
<u>APÊNDICE C - CLASSE PARALLELPORT</u>	120
<u>APÊNDICE D - CLASSE IEEE1451.1</u>	126



CAPÍTULO - 1

1. Introdução

Atualmente projetistas de sistemas de Medição e Controle Distribuído (DMC, do inglês: *Distributed Measurement and Control*) buscam a integração padronizada de transdutores (sensor/atuador) em rede de comunicação. O funcionamento de transdutores em rede através de interfaces padronizadas facilita a elaboração de projetos e aumenta o leque de aplicação da instrumentação distribuída, além de possibilitar a redução de custo de um DMC. Motivados para solucionar este desafio, o *National Institute of Standard and Tecnology* (NIST) e o *Institute of Eletrical and Eletronic Engineers* (IEEE) estabeleceram conceitos para construir um transdutor inteligente e, conseqüentemente, conectá-lo na rede de comunicação, dando origem ao denominado padrão IEEE1451 [1]. Para desenvolver o transdutor inteligente, o NIST e o IEEE propõem o emprego de ferramentas padronizadas, sistemas abertos, ferramentas de domínio público, sistema distribuído e linguagem de programação orientada a objeto. O padrão IEEE1451 é composto de diferentes comitês, sendo denominados de IEEE1451.0, IEEE1451.1, IEEE1451.2, IEEE1451.3, IEEE1451.4 e IEEE1451.5. Cada comitê é responsável por produzir as especificações necessárias para viabilizar a padronização de interfaces, na área de sistemas de DMC [2].

O comitê IEEE1451.0 tem como finalidade apresentar um modelo para implementar a interoperabilidade entre o NCAP (*Network Capable Application Processor*) e o acesso às TEDS (*Transducer Electronic Data Sheet*) [3].

O comitê IEEE1451.1 tem a função, de determinar as características funcionais de um processador com capacidade de operar em rede de comunicação NCAP [4].

O comitê IEEE1451.2 tem como objetivo, criar as diretrizes necessárias para desenvolver uma Interface Independente para Transdutores TII (*Transducer Independent Interface*) e um Módulo de Interface para Transdutores Inteligentes STIM (*Smart Transducer Interface Module*), o qual contém as informações de transdutores em formato eletrônico TEDS [5].



O comitê IEEE1451.3 é responsável por especificar uma interface de comunicação digital para sistemas distribuídos *multidrop*¹ [6].

O comitê IEEE1451.4 tem a responsabilidade de definir um protocolo de comunicação denominado de modo misto, isto é, para transmitir sinais analógicos e sinais digitais através do mesmo meio físico [7].

O comitê IEEE1451.5 deve desenvolver, uma interface padronizada para realizar a comunicação sem fio, mais conhecida como *wireless*, entre o NCAP e o STIM [8].

Na Figura 1.1 apresenta-se a configuração do padrão IEEE1451. Nesta figura, pode-se observar que o NCAP é uma interface comum para os demais padrões pertencentes ao IEEE1451.

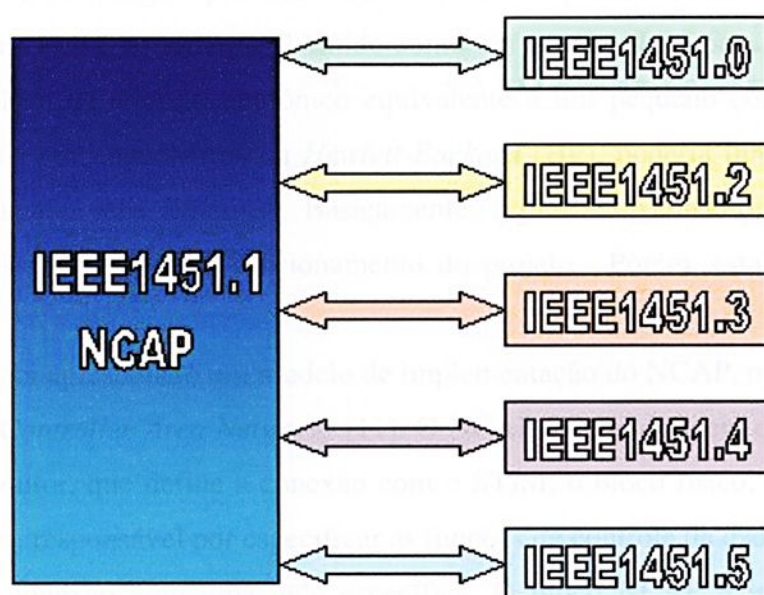


Figura 1.1: Configuração do padrão IEEE1451.

Os grupos de pesquisa em Sistemas Digitais e Instrumentação Eletrônica da FEIS/UNESP, Campus de Ilha Solteira, vêm estudando, desde 2001, a funcionalidade do padrão IEEE1451, especificamente, o IEEE1451.1 e o IEEE1451.2. Esta pesquisa deu origem ao tema de um projeto de dissertação de mestrado, para construir o NCAP, e um trabalho de doutorado²,

¹ *Multidrop*: configuração distribuída de vários transdutores do mesmo tipo, conectados a um barramento.

² O trabalho de doutorado, pertence ao doutorando Silvano Renato Rossi (UNESP-FEIS)

para implementar o STIM e uma interface que possibilita a construção do driver da TII, assim denominado de Gerenciador de Protocolo [9].

No modelo de informação IEEE1451.1 especificam-se que as atividades de um NCAP devem possuir interoperabilidade³ com as redes que controlam os transdutores inteligentes, e divide-se o NCAP em módulo hardware e módulo software.

A implementação de um NCAP foi tema de diversos trabalhos, alguns dos quais são discutidos nos parágrafos a seguir.

Em junho de 1999 foi publicado um artigo no qual descreve-se a estrutura para integração padronizada em DMC [10]. No artigo, apresentam-se conceitos do padrão IEEE1451, baseados em um projeto realizado no NIST. O NCAP é definido como um nó de rede e, para implementá-lo, foi construída uma placa de circuito eletrônico equivalente a um pequeno computador. A placa que foi desenvolvida por engenheiros da *Hewlett-Packard* (HP), poderia funcionar como uma rede DeviceNet⁴ ou uma rede Ethernet⁵. Basicamente, a placa foi construída utilizando microcontroladores, e possibilitou o bom funcionamento do projeto. Porém, esta placa não é comercializada.

Em maio de 2000 foi apresentado um modelo de implementação do NCAP, operando com protocolo de rede CAN (*Controller Area Network*) [11]. O NCAP foi dividido em quatro blocos básicos: o bloco de transdutor, que define a conexão com o STIM, o bloco físico, que define o NCAP, o bloco de funções, responsável por especificar as funções de controle de dados, e o bloco para rede, que define a conexão com uma rede específica. O bloco NCAP é implementado utilizando um microprocessador (μ C) de 16 bits, modelo 68HC916X1, da empresa Motorola. Conectado ao μ C está um barramento CAN, uma memória SRAM (*Static Random Access Memory*) opcional para implementar a TEDS, um pequeno circuito de memória flash e o STIM. O barramento CAN, composto pela CAN *controller* SJ1000 e pela CAN 82C250 é responsável por controlar o bloco de função do NCAP. O circuito de memória flash altera a tensão do sistema para implementar a leitura e escrita de dados.

³ Interoperabilidade: capacidade de funcionamento com diversos tipos de rede de comunicação.

⁴ DeviceNet: rede de comunicação desenvolvida pela Bosch para automação de automóveis, utilizando-se o protocolo CAN.

⁵ Ethernet: sistema de cabeamento que possibilita a comunicação em rede de dispositivos. A rede Ethernet será abordada no capítulo 3 deste texto.



Em abril de 2001, foi publicado um artigo no qual descreveu-se o desenvolvimento de um NCAP para cada sensor inteligente [12]. O NCAP foi implementado com o microprocessador da Rabbit⁶, que possibilita o desenvolvimento do modelo de informação IEEE1451.1. Este NCAP foi conectado na Intranet e possibilitou a interconectividade e acesso para a interface IEEE1451.2. O Java *Applet* foi utilizado para realizar a comunicação entre os usuários e os sensores, baseando-se no modelo de comunicação cliente/servidor.

Em julho de 2002, pesquisadores da Universidade Politécnica de Madri, apresentaram um projeto para construir um SoC (*System-on-Chip*) baseado no padrão IEEE 1451, para ser aplicado no interfaceamento de sistemas DMC [13]. No projeto, empregou-se a linguagem de descrição de hardware VHDL (*VHSIC Hardware Description Language*) na construção de um módulo que implementasse ambas as funções: de STIM e de NCAP. Este módulo foi sintetizado em FPGA (*Field Programmable Gate Array*) e conectado na porta serial do PC.

Em novembro de 2002, foi apresentada uma implementação do NCAP utilizando duas placas (EX01) e (EM04) [14]. A placa EX01 é um microcontrolador que permite a comunicação *TCP/IP*. Esta placa foi conectada em um computador pessoal (PC) através da porta paralela (DB-25). A placa EM04 controla a EX01, realizando a comunicação através de um *MUX* que seleciona serialmente uma interface *WebSensor*. A interface *WebSensor* foi projetada com base no padrão IEEE1451. Com esta interface foi possível a comunicação na Internet através da Ethernet.

Em maio 2003, pesquisadores do Instituto de Telecomunicações e da Escola Superior de Tecnologia, ambos de Portugal, utilizaram um PC e o LabView 6.1 para implementar o NCAP [15]. O Pc permitiu a construção da parte física do NCAP, além de conectá-lo na rede Ethernet. O LabView 6.1 ficou responsável por construir os módulos de softwares do NCAP. Uma interface intermediária foi construída para permitir ao LabView o controle da TII. Esta interface é uma biblioteca (*dll*), desenvolvida através da linguagem de programação C++, que tinha como objetivo implementar as funções necessárias para controlar o dispositivo conectado na porta paralela do PC.

Os diferentes modelos de NCAP devem-se às várias ferramentas que podem ser utilizadas no desenvolvimento do processador (hardware e software) e ao tipo de aplicação (linha de produção, controle da asa de um avião, telemedicina, etc.). Para implementar um NCAP, os

⁶ Rabbit Semiconductor empresa especializada na produção de microcontroladores.



projetistas devem estar atentos, aos conceitos de redes de comunicação, controle de interfaces⁷ e a utilização de uma linguagem de programação orientada a objeto.

No projeto apresentado nesta dissertação emprega-se, basicamente, o recurso de um PC, para construir os módulos hardware, e a linguagem de programação Java⁸, para construir as partes lógicas de um NCAP, além de utilizar um Gerenciador de Protocolo. O Gerenciador de Protocolo foi desenvolvido em linguagem de descrição de hardware *VHSIC Hardware Description Language* (VHDL) e, posteriormente, sintetizado em dispositivo do tipo FPGA (*Field Programmable Gate Array*). O Gerenciador de Protocolo, que faz parte de um trabalho de doutorado⁹, é responsável por executar o protocolo de comunicação entre o NCAP e o STIM.

O NCAP desenvolvido possibilita a execução de atividades remotas de um dispositivo FPGA, no qual podem ser conectados diferentes tipos de transdutores.

A parte lógica do NCAP é implementada através do desenvolvimento de um software, aqui denominado de software NCAP. Este software emprega a arquitetura de comunicação cliente/servidor, pertencente à tecnologia Java, e é composto por três programas: *IEEE1451_BaseTransducerBlock*, *IEEE1451_NCAPBlock* e *IEEE1451_FunctionBlock*. Com a execução desses três programas, é possível que clientes conectados em um ponto de uma rede local ou a Internet, envie endereços de funções para o servidor localizado em outro ponto da rede. Através da porta paralela de um PC, o servidor transfere os endereços de funções ao Gerenciador de Protocolo. A utilização da linguagem Java, para desenvolver o software NCAP, justifica-se pelas seguintes características desta tecnologia:

- a) É orientada a Objeto, OO [16]: No modelo de informação responsável por especificar as características de um NCAP, propõe-se a utilização de linguagens OO para desenvolver os módulos softwares do NCAP;
- b) A linguagem Java vem sendo empregada em diversas áreas da engenharia, como: tv digital, celular, robótica, laboratórios virtuais e sistemas embarcados;

⁷Interface: neste caso, hardware que pode desenvolver atividades como, por exemplo, microcontrolador, FPGA, etc .

⁸A linguagem de Programação Java pertence à Sun Microsystems

⁹O Gerenciador de Protocolo pertence ao trabalho do doutorando Silvano Renato Rossi (UNESP-FEIS)



- c) O compilador e os pacotes Java podem ser obtidos gratuitamente no site da Sun Microsystems: www.java.sun;
- d) As aplicações Java são portáteis, característica importantíssima, pois, permite que um mesmo programa seja executado em plataformas¹⁰ diferentes;
- e) A linguagem Java pode ser utilizada em diferentes níveis de interfaceamento (interface gráfica e controle de hardware).

Com a finalidade de ser utilizado em diversas aplicações e obter maior qualidade, o software para NCAP, aqui desenvolvido, é um software livre. Assim, este software possibilita a redução de custo e a flexibilidade na manutenção de um sistema de monitoramento remoto.

Neste projeto, o NCAP utiliza a rede Ethernet para realizar a comunicação entre clientes e servidor. Desta forma, a rede local atua como barramento de campo¹¹ e como rede de controle¹².

A Ethernet é uma tendência para as redes de comunicações industriais, por ser uma rede padronizada, de baixo custo, que vem obtendo grandes evoluções nos sistemas de cabeamentos, que proporciona transmissão de dados, imagens e sons em alta velocidade, que possibilita a inserção de diversos protocolos de barramento de campo para tornar a rede interoperável.

Assim, este trabalho de pesquisa emprega uma tecnologia emergente para construir um NCAP em conformidade com o padrão IEEE1451.

No capítulo 2 descreve-se os conceitos fundamentais para implementar uma rede de comunicação, baseando-se nas redes de computadores.

No capítulo 3 é apresentada-se as características de uma rede local, destaque para a rede Ethernet, empregada no projeto.

O capítulo 4 fornece as principais características de funcionamento da tecnologia Java, empregadas no projeto.

No capítulo 5 descreve-se o funcionamento do padrão IEEE1451.2 e IEEE1451.1.

No capítulo 6 mostra-se o desenvolvimento do software para implementar a parte lógica do NCAP e os testes realizados com a execução deste software.

No capítulo 7 descreve-se as conclusões sobre o trabalho.

¹⁰ plataformas: refere-se aos sistemas operacionais.

¹¹ barramento de campo: rede de comunicação utilizada para conectar transdutores.

¹² Rede de controle: realiza a comunicação entre o barramento de campo e os usuários.



2. Rede de computadores

2.1 Introdução

Os sistemas de medição e controle distribuído (DMC) estão intimamente ligados aos funcionamentos de uma rede de computadores (ou rede de comunicação). Nos conceitos elaborados pelo padrão IEEE1451, os transdutores devem funcionar como um computador conectado na rede, por isso, neste capítulo, apresentam-se os principais conceitos relacionados a uma rede de computador, com a finalidade de apresentar e aplicar tais conceitos no desenvolvimento de uma rede de transdutores inteligentes.

Define-se uma rede de comunicação como sendo um sistema constituído por vários hardware trocando informações e compartilhando tarefas [17].

Uma rede de comunicação aplicada na indústria, por exemplo, pode ser dividida em três níveis: superior, intermediário e inferior. O nível superior é onde se encontra o cliente ou usuário que realiza a atividade de supervisão e controle. Este nível é, basicamente, formado por um PC, em termos de hardware, enquanto que na parte de software, o Java vem sendo muito utilizado.

O nível intermediário é onde se encontram as redes de controle, e faz a interface entre os barramentos de campo e o supervisor. O nível inferior é onde se localiza o barramento de campo. Este nível é responsável por conectar sensores, atuadores, microcomputadores, etc.

As comunicações dentro de cada nível ou entre os níveis foram divididas em camadas. Cada camada é responsável por um serviço específico na comunicação e está relacionada com a camada correspondente de outra máquina (computador), ou seja, a camada *n* de uma máquina se comunica com a camada *n* de outra máquina. Como as camadas possuem serviços diferentes, a organização desses serviços é chamada de protocolo. O conjunto de protocolos é chamado de arquitetura de rede.



A finalidade da arquitetura de rede é dar suporte suficiente para desenvolver software e/ou hardware para que haja comunicação entre as máquinas desta rede. As instituições *International Standards Organization* (ISO) e IEEE criaram o modelo de referência ISO-OSI (*Open System Interconnection*), com o objetivo de produzir padrões de conectividade para interligar computadores de diferentes fabricantes. Este fato torna o modelo atraente para se aplicar no desenvolvimento do padrão IEEE1451. O Modelo de Referência, denominado de RM-OSI (*Model Reference*), é uma arquitetura de rede, porém, não especifica os serviços e os protocolos que devem ser usados em cada camada, mas apenas informa o que cada camada deve fazer. O RM-OSI apresenta aspectos gerais da rede que estão divididos em sete camadas funcionais, facilitando assim a compreensão de questões fundamentais sobre uma rede. Na Figura 2.1 mostra-se a arquitetura do modelo RM-OSI.

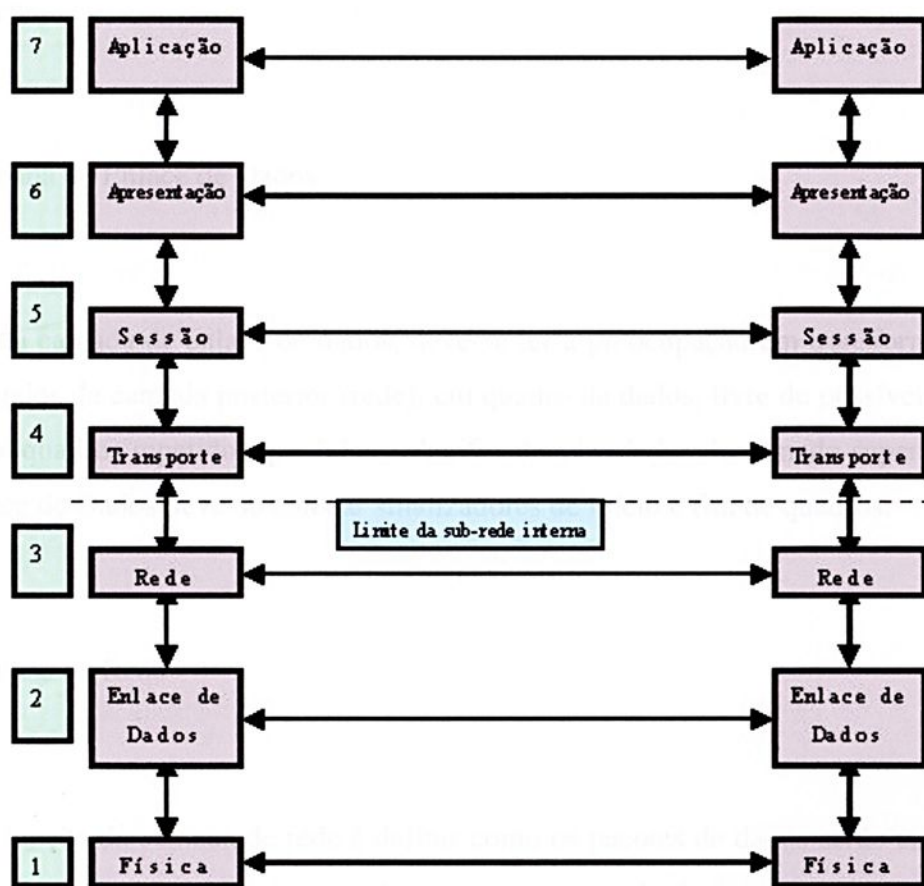


Figura 2.1: O modelo de Referência RM-OSI.

2.2 Camadas pertencentes ao RM-OSI

2.2.1 A Camada Física

Como o nome já indica, a camada física está relacionada com a parte de ligação física entre as máquinas. Nesta camada é feito o tratamento dos bits enviados/recebidos através de um canal de comunicação (por exemplo: sistema de cabeamento), sendo definidos parâmetro como o tempo de transmissão, o valor da tensão relacionada aos bits 0 e 1, o modo de transmissão e a estrutura das pinagens.

2.2.2 A Camada de Enlace de Dados

No projeto da camada de enlace de dados, deve-se ter a preocupação em transformar um pacote de dados, vindos da camada posterior (rede), em quadro de dados, livre de possíveis erros como, por exemplo, quadros repetidos, perdidos e danificados, herdados da camada superior etc. Na camada de Enlace de Dados deve-se colocar sinalizadores de início e fim de quadros.

2.2.3 A Camada de Rede

A principal função da camada de rede é definir como os pacotes de dados serão enviados da origem para o destino. Compete à esta camada executar o controle de congestionamento para evitar um engarrafamento de dados. Quando um quadro de dados vai de uma rede (origem) para outra (destino), podem surgir muitos problemas como, por exemplo, o endereçamento utilizado pelas redes serem diferentes, a rede destino não aceitar o pacote devido, o seu tamanho e os protocolos serem diferentes.



2.2.4 A Camada de Transporte

A função da camada de transporte é a de aceitar dados da camada posterior (sessão), dividi-los em pequenas unidades, quando necessário, passá-los à camada de rede e assegurar que todas estas unidades de dados cheguem corretamente à outra extremidade da conexão. Tudo isto, deverá ser feito de uma forma eficiente e de modo que a camada de sessão seja imune às inevitáveis mudanças de tecnologia de hardware.

2.2.5 A Camada de Sessão

A camada de sessão permite que os usuários em diferentes máquinas estabeleçam sessões entre eles. Um dos serviços dessa camada é o de gerenciar o controle do “diálogo” entre máquinas.

2.2.6 A Camada de Apresentação

Ao contrário das camadas inferiores, que estão relacionadas com o transporte de bits de uma forma eficiente de um lado para o outro, a camada de apresentação preocupa-se com a estrutura da informação transmitida. A camada de apresentação gerencia essas estruturas de dados abstratas e converte a representação utilizada dentro do computador na representação padrão da rede, e vice-versa.



2.2.7 A Camada de Aplicação

A camada de aplicação pode conter uma variedade de protocolos necessários para implementar a interface com o usuário. Nesta camada são definidas as funções de gerenciamento e mecanismos que servem para a construção de aplicações distribuídas.

As redes de comunicação podem ser divididas conforme a atuação geográfica e são classificadas como sendo: *Local Area Network* (LAN), *Wide Area Network* (MAN) e *World Area Network* (WAN).

As redes locais ou LAN, tradicionalmente, possuem até 3 Km de diâmetro e cobrem, geralmente, uma área localizada. Contidas em um prédio, em um campus universitário ou na linha de montagem de uma fábrica, a tecnologia de transmissão das redes locais quase sempre consiste em um cabo no qual todas as máquinas são conectadas. Neste tipo de rede encontram-se os barramentos de campo, nos quais devem ser conectados os transdutores.

As redes metropolitanas ou MAN são, na verdade, uma versão ampliada das redes locais e cobrem áreas superiores a 50 Km de diâmetro. Finalmente, as redes geograficamente distribuídas ou WAN abrangem geograficamente toda a área do planeta. Têm facilidade de comunicação em nível nacional e internacional e podem ligar várias LAN's ou MAN's, além de suportarem diversas velocidades de transmissão de dados à rede mundial de computadores. A Internet é um exemplo de WAN.

Pelo fato das redes LANs constituírem a base dos sistemas de comunicação de transdutores em rede, é conveniente adquirir conceitos sobre o funcionamento desta tecnologia para poder aumentar o desempenho da comunicação. As redes locais podem ser divididas de acordo com a topologia: em anel, barramento e estrela.

2.3 Topologia de uma rede de comunicação



Os diferentes tipos de topologia, são opções para que os projetistas possam implementar redes de comunicação (transdutores e computadores), de acordo com as necessidades de desempenho, custo, flexibilidade, etc.

2.3.1 Topologia em Anel

Na topologia em anel todos os dispositivos são ligados a um sistema de cabeamento disposto em círculo ou anel, conforme representado na Figura 2.2. Este tipo de rede utiliza uma série de pacotes vazios, chamados *tokens*, que estão sempre circulando na rede. Uma estação de origem, simplesmente, retira um pacote da rede, examina seu conteúdo e, se estiver vazio, introduz os dados necessários para a transmissão. Por outro lado, se o pacote estiver cheio, o sistema operacional da rede o devolve e seleciona outro. O endereço de destino e a seqüência de identificação do pacote são afixados ao *token* antes que ele retorne à rede para entrega.

A vantagem de utilizar a tecnologia *token* é que a colisão de dados fica eliminada e a velocidade de transmissão efetiva é muito alta, mesmo em rede de grande tráfego. Porém, o custo dos cabos e das placas de rede para topologias *Token Ring* são considerados elevados.

Outra desvantagem é que neste tipo de rede, para haver um correto funcionamento, o *token* tem sempre que dar uma volta completa e, se houver qualquer problema com o cabeamento, toda a rede sai de operação.

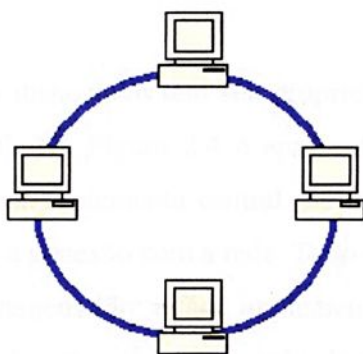


Figura 2.2: Topologia em anel.

2.3.2 Topologia em Barramento

Na topologia em barramento os dispositivos estão conectados em uma fila ao longo de um único cabo (segmento), como é ilustrado na Figura 2.3. Os dispositivos em barramento comunicam-se endereçando os dados a um computador em particular e inserindo estes dados no cabo. As vantagens estão associadas ao uso de cabos com economia, ter simplicidade, confiabilidade e facilidade de ampliação. Como desvantagens, cita-se que a rede pode ficar lenta diante de um tráfego intenso, que apresenta problemas difíceis de serem isolados e que os possíveis rompimentos dos cabos podem afetar muitos usuários.

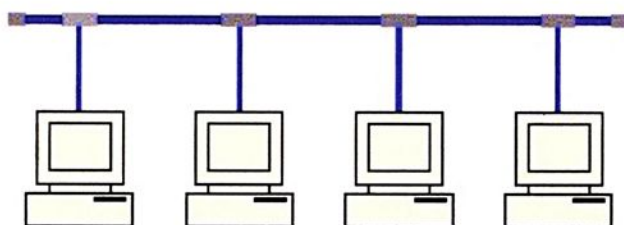


Figura 2.3: Topologia em Barramento.

2.3.3 Topologia Estrela

Na topologia em estrela cada dispositivo tem seu próprio cabo partindo do gabinete de fiação central (concentrador ou *hub*). Na Figura 2.4 é apresentada uma configuração para a topologia em estrela, com um *hub* como elemento central. Se um fio se romper ou entrar em curto-circuito, somente esse nó perde a conexão com a rede. Todo o tráfego tem obrigatoriamente que passar pelo concentrador. As vantagens são: a fácil implementação técnica e os bons tempos de resposta pelos terminais. A desvantagem está na utilização de um equipamento dedicado, que causa um aumento no custo desta configuração.

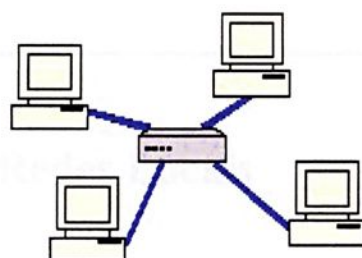


Figura 2.4: Topologia Estrela.

3. Padronização das Redes Locais

3.1 Introdução

O IEEE produziu vários padrões para as Redes Locais. Estes padrões são comumente conhecidos por **IEEE 802**, e incluem as seguintes tecnologias: *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD), *Token bus* e *Token ring*. Os vários padrões diferem, principalmente, no nível físico e no sub-nível *Medium Access Control* (MAC), como mostra a Figura 3.1 [18].

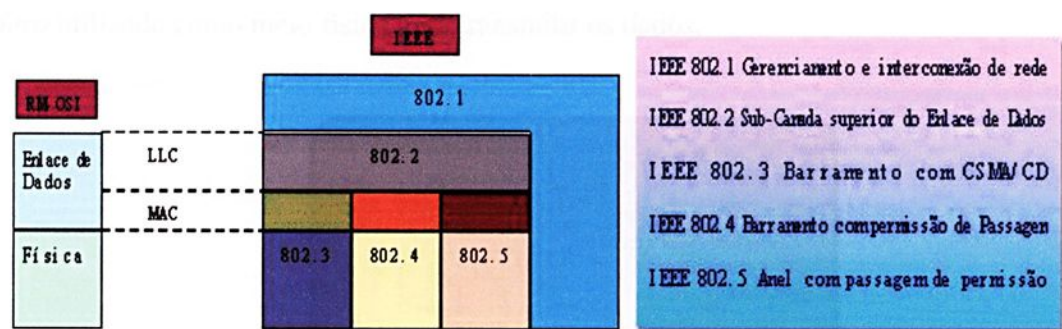


Figura 3.1: Modelo de arquitetura IEEE 802.

Os padrões estão divididos em partes, e cada um deles publicado em livros específicos. O padrão 802.1 fornece uma introdução ao conjunto de padrões e define as primitivas da interface.

O padrão 802.2 descreve a parte superior do nível de ligação de dados, que utiliza o Controle de Enlace Lógico, o *Logic Link Control* (LLC). Os padrões 802.3 até 802.5 descrevem os três padrões de redes locais.



O IEEE e o NIST vêm utilizando a rede Ethernet (correspondente ao padrão 802.3) para desenvolver projetos em conformidade com o padrão IEEE1451.

A utilização de uma rede local padronizada é viável para construir sistemas de instrumentação inteligente em conformidade com o padrão IEEE1451, pois esta tecnologia está presente em grande parte das atividades relacionadas com a Internet. Dentro deste cenário, destaca-se a rede Ethernet.

As motivações para empregar a rede Ethernet na troca de informações entre os transdutores inteligentes e a Internet são: a popularidade, o baixo custo, a flexibilidade e o fato de ser uma rede aberta e padronizada.

3.2 O padrão IEEE 802.3

O padrão Ethernet foi um dos pioneiros em arquitetura de rede local. A Figura 3.2 mostra o primeiro rascunho produzido para este fim. O nome Ethernet é uma referência ao canal *éter luminoso* utilizado como meio físico para transmitir os dados.

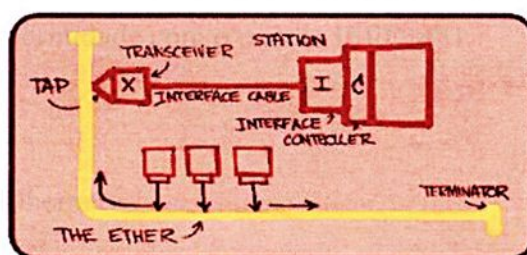


Figura 3.2: Rascunho da ethernet. fonte: <http://www.ieee802.org/3/>

A Ethernet é um esquema de cabeamento e sinalização para redes locais que chegou ao mercado no final dos anos 70 e continua sendo um padrão utilizado até hoje. A razão para a longa vida do padrão Ethernet é simples: ele proporciona transmissão de alta velocidade a um preço econômico, representa uma base sólida para implementações de diversas aplicações de rede locais e possui a importante característica de ser um sistema aberto.

A Ethernet foi criada em 1972 pelo engenheiro eletricista *Robert Metcalfe* [19], quando trabalhava na Xerox, no lendário PARC (*Palo Alto Research Center*). Com a rede Ethernet, a Xerox possibilitou que todas as estações compartilhassem do mesmo meio de transmissão. No início, esta tecnologia foi denominada de “*Network Alto Aloha*”.

A Xerox acrescentou a detecção de portadora, criando o sistema CSMA/CD de 2,94 Mbps para serem conectadas a 100 estações de trabalho pessoais. A rede possuía um cabo coaxial de 1 km e a configuração utilizada, para esta conexão, foi a de barramento.

A Ethernet da Xerox foi tão bem sucedida que ela, a DEC e a Intel criaram um padrão para um sistema Ethernet de 10 Mbps. Já em 1976, *Metcalfe* e *Boggs* publicaram um trabalho sobre a Ethernet, cujas especificações técnicas podiam ser aplicadas em máquinas de diferentes fabricantes, tornando a Ethernet uma “rede aberta” [19]. Por ser um dos primeiros sistemas abertos de comunicação, em uma época onde a falta de padrão dificultava o progresso das pesquisas e a venda de equipamentos, o padrão Ethernet foi adotado pelo IEEE.

Assim, o padrão Ethernet formou a base do IEEE 802.3. O padrão 802.3 descreve uma família de sistemas CSMA/CD que funcionava em velocidades entre 1 e 10 Mbps em diversos meios. A Ethernet é atualmente a rede local de maior popularidade e, conseqüentemente, uma rede de baixo custo. Desta forma, a Ethernet é uma excelente opção de barramento de campo para desenvolver a comunicação em conformidade com o padrão IEEE1451.

3.3 Especificações da rede Ethernet

3.3.1 Modo de Transmissão

A transmissão dos dados em uma rede Ethernet pode ser realizada das seguintes maneiras:

- a) *Simplex*: A transmissão é unilateral, ou seja, só um nó transmite os dados;
- b) *Half-duplex*: cada nó transmite ou recebe informações, não acontecendo transmissão simultânea;



- c) *Full-duplex*: cada nó transmite e/ou recebe, podendo ocorrer transmissão simultânea.

3.3.2 Controle de Acesso ao Meio

a) A subcamada MAC 802.3

As séries de padrões IEEE 802 dividem a segunda camada (Enlace de dados) do modelo de RM-OSI, em duas subcamadas: a subcamada inferior é chamada de *medium access control* (MAC), ou seja, controle de acesso ao meio, enquanto que a subcamada superior é chamada de *logical link control* (LLC), ou seja, controle de enlace lógico. O controle de acesso ao meio determina os procedimentos que a rede de comunicação deve realizar para inserir os dados no cabo de transmissão. A especificação Ethernet IEEE802.3 envolve desde a camada Física até a subcamada MAC. Desta forma, quando um nó transmite um quadro de dados, a subcamada MAC monta o endereço de destino e os endereços de origem para cada quadro. Além disso, a detecção de colisões, o tratamento de protocolos e o algoritmo CSMA/CD também acontecem no nível da subcamada MAC. Na Figura 3.3 mostram-se os estados de um quadro Ethernet.

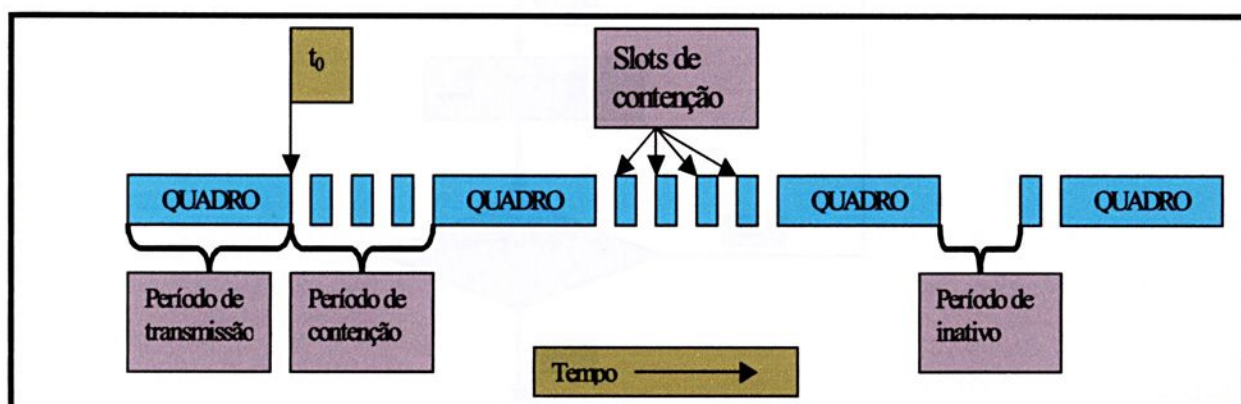


Figura 3.3: Estados de um quadro Ethernet.

b) Algoritmo CSMA/CD

O padrão IEEE 802.3 utiliza o protocolo CSMA/CD para coordenar o acesso ao meio de transmissão. No protocolo CSMA/CD, quando um nó quer transmitir, primeiramente, analisa o estado do cabo. Se o cabo estiver ocupado o nó aguarda até ele ficar livre, caso contrário, o nó começa imediatamente a transmissão. Se dois ou mais nós começarem a transmitir simultaneamente em um cabo desocupado, ao invés de finalizar a transmissão de seus quadros, eles devem interromper a transmissão abruptamente após a colisão detectada. Neste caso, os nós aguardam um tempo aleatório e repetem todo o processo novamente. Este protocolo é amplamente usado na sub-camada MAC de LANs. O fluxograma de funcionamento do protocolo CSMA/CD é ilustrado na Figura 3.4.

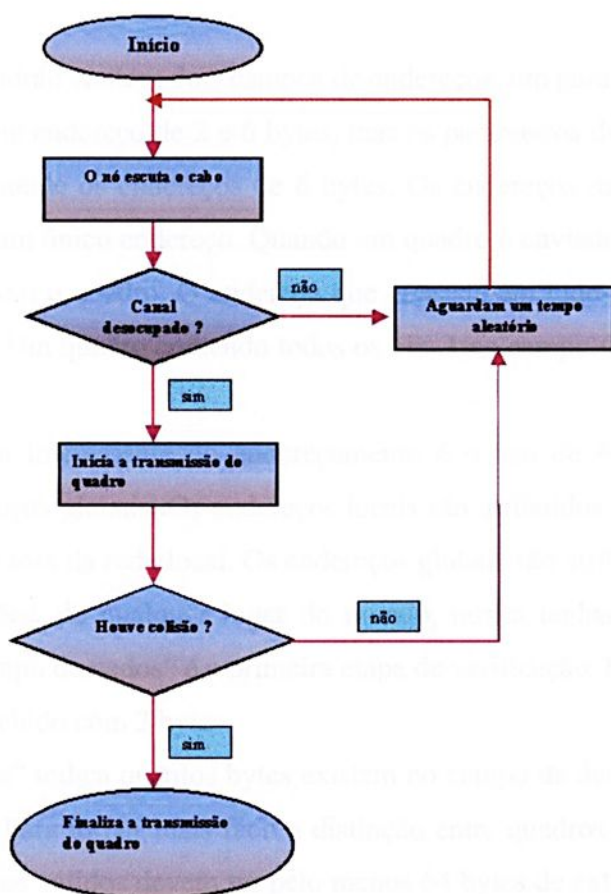


Figura 3.4: Funcionamento do CSMA/CD.

c) Padronização do Quadro de Bits

A estrutura do quadro 802.3 é apresentada na Figura 3.5. Cada quadro começa com um “preâmbulo” de 7 bytes, cada um contendo o padrão de bit “10101010”, que são usados para sincronização de bit. Em seguida vem o byte chamado de “início de quadro”, que é responsável por sincronizar o início do quadro propriamente dito.

Preâmbulo 7 bytes	Início 1 byte	Endereço de destino 6 bytes	Endereço de origem 6 bytes	Tamanho do campo de dados 2 bytes	Dados 0-1500 bytes	Enchimento 0-46 bytes	Soma Verificação 4 bytes
-----------------------------	-------------------------	---------------------------------------	--------------------------------------	---	------------------------------	---------------------------------	------------------------------------

Figura 3.5: Padrão de formação de um quadro de bits (802.3).

O quadro de bits padrão contém dois campos de endereços, um para “destino” e um para a “origem”. O padrão permite endereço de 2 e 6 bytes, mas os parâmetros definidos para o padrão de banda básica usam somente os endereços de 6 bytes. Os endereços de grupo permitem que diversos nós reconheçam um único endereço. Quando um quadro é enviado para um endereço de grupo, todos os nós recebem o quadro. O endereço que consiste em todos os bits 1 é reservado para difusão (*broad-cast*). Um quadro contendo todos os bits 1 no campo de destino é entregue a todos os nós da rede.

Outra característica interessante do endereçamento é o uso de 46 bits para distinguir endereços locais de endereços globais. Os endereços locais são atribuídos pelo administrador de rede e não têm significado fora da rede local. Os endereços globais são atribuídos pelo IEEE para assegurar que duas estações, de qualquer lugar do mundo, nunca tenham o mesmo endereço global. O “tamanho do campo de dados” é a primeira etapa de verificação. Se o dado está correto, este campo deve ser preenchido com 2 bytes.

O campo de “dados” indica quantos bytes existem no campo de dados, de um mínimo de 0 a um máximo de 1500. Para tornar mais fácil a distinção entre quadros válidos e inválidos, o 802.3 afirma que os quadros válidos devem ter pelo menos 64 bytes de extensão, do endereço de destino até o *checksum* (Soma de verificação).

A razão para se ter um quadro de comprimento mínimo é evitar que uma estação conclua a transmissão de um quadro curto, antes de o primeiro bit ter atingido a extremidade do cabo, onde ele pode colidir com um outro quadro. O “enchimento” existe para garantir que um quadro tenha um tamanho mínimo para ser válido. Caso o dado seja menor que 46 bytes este campo se encarrega de fornecer a quantidade necessária para montar um quadro válido.

O campo final do quadro 802.3 é a “soma de verificação”, o qual trata-se, efetivamente, de um código de verificação de dados de 32 bits. Se alguns bits de dados estiverem sendo recebidos com erro (devido a ruído no cabo), a soma de verificação certamente estará errada, e o erro será detectado. O problema de colisão é ilustrado na Figura 3.6. Com base nesta figura tem-se: no tempo t_0 , um nó A , em uma extremidade da rede, envia um quadro ao nó B . Antes do quadro chegar à outra extremidade (B), o nó B também começa a transmitir. Quando o nó B detecta um aumento anormal na potência útil para transmitir, evidencia-se que ocorreu uma colisão. Neste instante, o nó B interrompe a transmissão e gera uma rajada de ruído de 48 bits para avisar todas as estações. O nó A recebe a saída do ruído e também interrompe a transmissão. Em seguida, os nós A e B aguardam um intervalo de tempo aleatório antes de iniciar novamente a segunda tentativa de transmissão.

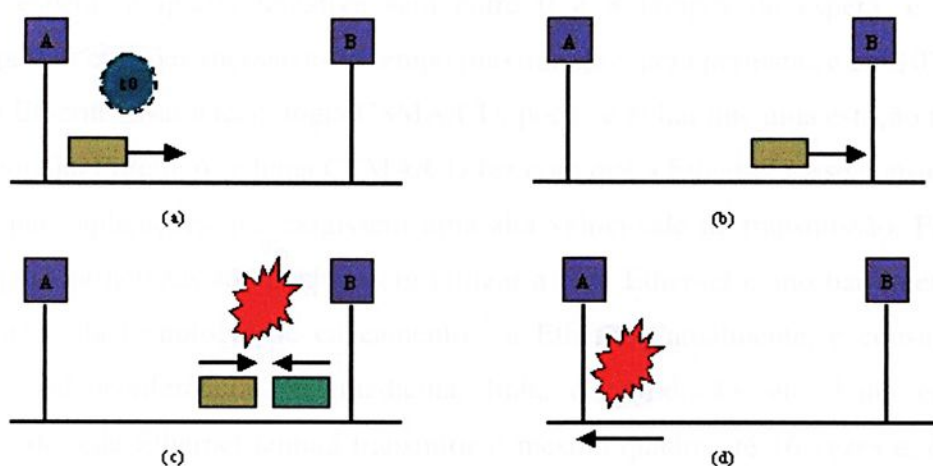


Figura 3.6: Detecção de colisão.

- (a) O nó A envia um quadro ao nó B , (b) O quadro aproxima-se do destino
 (b) O nó B começa a transmitir (d) Rajada de ruído atinge o nó A
 indicando a ocorrência de colisão.

d) Algoritmo de Backoff

O algoritmo de *Backoff* é uma solução empregada para o tratamento de colisões repetidas na rede Ethernet, que requer que cada estação afetada por uma colisão espere uma quantidade de tempo aleatório antes de voltar a transmitir. A estação que escolheu o tempo aleatório mais curto estará pronta para transmitir o seu quadro, sem a interferência da estação que contribuiu para a colisão anterior. Uma vez reconhecida a retransmissão da estação que escolheu o tempo aleatório mais curto dentre todas as estações da rede, todas devem esperar que a transmissão se complete antes de novamente competirem pelo acesso à rede.

Cada vez que uma estação tenta retransmitir e encontra uma colisão, o tempo máximo de espera é duplicado. Se a primeira tentativa de transmissão falhar, devido a uma colisão, todas as estações envolvidas na colisão aguardarão entre 0 e 2 de tempos de espera¹³, antes de voltarem a tentar transmitir o quadro uma segunda vez. Em uma Ethernet a 10 Mbps, isto significa que o tempo de espera será de $0 \times 51,2$ micro segundos, de $1 \times 51,2$ micro segundos ou de $2 \times 51,2$ micro segundos. À terceira tentativa de transmitir o quadro, o tempo máximo de espera será entre 0 e 4 tempos de espera; à quarta tentativa será entre 0 e 8 tempos de espera, e assim sucessivamente. Após 11 colisões sucessivas, o tempo máximo de espera permanece em 1023.

Pelo fato da Ethernet usar a tecnologia CSMA/CD, pode-se evitar que uma estação receba a informação deteriorada. Porém, o sistema CSMA/CD fez com que a Ethernet fosse considerada não determinística para aplicações que exigissem uma alta velocidade de transmissão. Foi por este motivo que alguns projetistas se opunham em utilizar a rede Ethernet como barramento de campo. Com o avanço da tecnologia de cabeamento¹⁴ a Ethernet, atualmente, é considerada determinística para videoconferências, telemedicina, linha de produção, etc. Uma estação contendo uma placa de rede Ethernet tentará transmitir o mesmo quadro até 16 vezes e, depois disso, se a transmissão não tiver sucesso, desiste e descarta o quadro. Na Tabela 3.1 apresenta-se os tempos de espera de cada estação para as tentativas de transmissão de informação.

¹³ Tempo de espera ou slot time é o tempo randômico que a estação pode aguardar para tentar transmitir um dado.

¹⁴ Cabeamento: é apresentado na seção 3.3.3 Meio Físico.



Tabela 3.1: Tentativa de transmissão na rede Ethernet.

Tentativa de Transmissão	Tempo Máximo de espera
1	N/A
2	2
3	4
4	8
5	16
6	32
7	64
8	128
9	256
10	512
11	1023
12	1023
13	1023
14	1023
15	1023
16	1023
17	DESISTE

3.3.3 Meio Físico

a) Ethernet 10 Mbps

A Ethernet utiliza vários tipos de cabeamento, resultando em diferentes valores na taxa de transmissão. Os vários tipos de cabeamento são definidos no Meio Físico. O Meio Físico define o tipo de cabo e suas características. A primeira família de cabos que foram padronizados pelo IEEE é apresentada na Tabela 3.2. O padrão IEEE802.3 fornece os parâmetros para um sistema de *banda básica*¹⁵ de 10 Mbps (10Base). Quatro tipos de cabos são comumente usados para a transmissão de 10 Mbps conforme apresentado na Tabela 3.2.

Tabela 3.1: Cabeamento para 10Mbps no padrão 802.3.

Parâmetro	10Base 5	10Base 2	10Base T	10Base F
Tipo de Cabos	Coaxial grosso	Coaxial fino	Par trançado	Fibra óptica
Topologia	Barramento	Barramento	Estrela (hub)	
Conector utilizado	Transceptor + cabo AUI	Conector T + BNC	RJ-45	RJ-45
Slot máximo	500 m	200 m	100m	2000 m
Comprimento Máximo da Rede	500 m	925m	2500	
Nós por segmentos	100	30	1024	1024
Distância mínima entre dois nós	2,5m	0,5m	Não há	
Vantagens	Sistema mais barato	Ideal backbone	Fácil manutenção	Melhor para edifícios
Velocidade	10 Mbps	10 Mbps	10 Mbps	10 Mbps

¹⁵ Banda básica: *baseband* corresponde a faixa transmissão de sinais digitais não modulados. Neste caso “base” aceita somente quadros Ethernet.



A notação 10Base5 significa que opera-se a 10 Mbps, utiliza-se a sinalização de *banda básica* e pode-se aceitar que um dispositivo fique a uma distância da estação de controle por até 500 metros. O cabeamento 10Base5 é popularmente chamado de Ethernet grosso, o qual foi o primeiro cabo a ser desenvolvido, possuindo uma capacidade de controlar até 100 dispositivos. O Ethernet grosso se parece com uma mangueira de jardim e possui quatro pares trançados blindados individualmente, como mostrado na Figura 3.7.

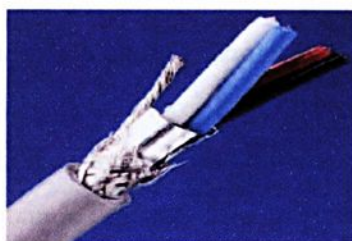
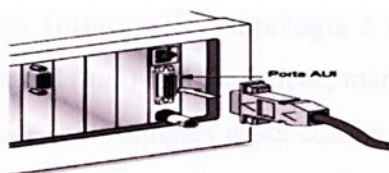


Figura 3.7: Cabo coaxial grosso. Fonte: <http://www.clubedohardware.com.br/aula4.html>

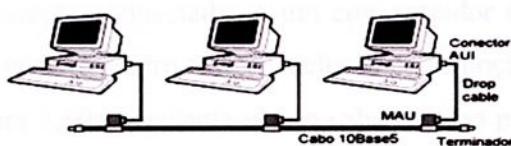
Dois dos pares são destinados à entrada e saída de dados, e os outros dois pares são destinados a sinais de controle da entrada e saída. Geralmente, as conexões com Ethernet grosso são feitas com conectores de pressão, nos quais um pino é cuidadosamente inserido até a metade na parte central do cabo coaxial. Além disso, um transceptor comumente chamado de “vampiro” é preso firmemente ao cabo, para que seu conector de pressão faça contato com o núcleo interno do cabo. O 10Base5, utiliza o conector AUI¹⁶ para conectar-se na placa de rede e é ligado através de um cabo drop e a um *Transceiver* denominado de MAU¹⁷. Nas Figuras 3.8a, 3.8b e apresenta-se a tecnologia 10Base5, sendo que na Figura 3.8a mostra-se a porta AUI da placa de rede e, na Figura 3.8b, o esquema para usar o 10Base5. As terminações que aparecem na Figura 3.8b servem para evitar a reflexão de sinais.

¹⁶ Attachment Unit Interface (AUI)

¹⁷ Media Attachment Unit (MAU), ou Media Access Unit, ou ainda, Multistation Access Unit



(a)



(b)

Figura 3.8: Tecnologia 10Base5.

(a) Ligação no conector AUI e (b) Rede de computadores interligados

Fonte: <http://www.laercio.com.br/site2/artigos/hardware/hard-020/hard-020.htm>

O cabeamento 10Base2 é conhecido como Ethernet fino. Em vez do uso de conectores de pressão, as conexões são feitas através de conectores BNC¹⁸. Neste caso o transceptor fica junto à placa de rede. O 10Base2 é adequado para locais com pouca interferência eletromagnética, sendo bem mais barato e fácil de instalar quando comparado com os demais cabeamentos, porém, pode abranger apenas 200 metros e só é capaz de controlar 30 dispositivos. Na Figura 3.9a mostra os conectores e terminações utilizados para o 10Base2. Na Figura 3.9b, mostra-se o Ethernet Fino, e na Figura 3.9c, mostra-se uma conexão empregando a tecnologia 10Base2.



(a)



(b)



(c)

Figura 3.9: Configuração para o 10Base2.

(a) Conector BNC, T e terminador, (b) Ethernet Fino e (c) Ethernet Fino em uma placa de rede, utilizando conector "T".

Fonte: <http://www.laercio.com.br/site2/artigos/hardware/hard-020/hard-020.htm>.

¹⁸ BNC: Bayonet-Neil-Concelman. Neil e Concelman são os nomes dos engenheiros que conduziram o desenvolvimento deste conector. e Bayonet (traduz-se como Baioneta).

No 10Base T¹⁹, a topologia é composta por estações conectadas a um concentrador de cabos central (*hub*) que estabelece, mantém ou corta a conexão entre elas. A inclusão ou remoção de um nó é mais simples nesta configuração. Na Figura 3.10a apresenta-se um cabo do tipo par trançado com conector RJ-45, utilizado no sistema 10Base T. Na Figura 3.10b ilustra-se uma configuração utilizando a tecnologia 10Base T. Nesta configuração, vários nós comunicam-se através de um *hub*.



Figura 3.10: Tecnologia 10Base T.

(a) Cabo par trançado, com RJ-45 (b) Conexão 10Base T

Fonte: <http://www.laercio.com.br/site2/artigos/hardware/hard-020/hard-020.htm>.

O sistema 10base-T apresenta um esquema para detecção de defeitos ao longo do cabo. Neste esquema, a detecção de cabos partidos, conectores defeituosos ou conectores soltos podem ser detectados, basicamente, ao injetar um pulso no cabo. Se o pulso atingir um obstáculo ou o fim do quadro, um eco será gerado e enviado de volta. Cronometrando-se, cuidadosamente, o intervalo entre o envio do pulso e a recepção do eco, é possível localizar sua origem.

A especificação 10Base-F utiliza a fibra ótica como meio de transmissão. Na Figura 3.11 apresenta-se um cabo de fibra ótica. Apesar de ser uma alternativa cara, em função do custo dos conectores e terminais, apresenta uma excelente imunidade a ruídos, o que a torna o meio ideal para interligar segmentos da rede localizados em prédios diferentes.

¹⁹ O “T” se refere ao termo twisted, ou seja, uma referência ao par trançado utilizado no sistema 10Base T.

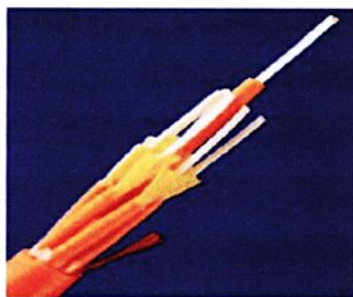


Figura 3.11: Foto de uma fibra óptica.fonte:

<http://www.clubedohardware.com.br/aula4.html>

As vantagens de um sistema de comunicação utilizar cabos a fibra óptica são:

- a) Maior velocidade;
- b) Maior isolamento elétrico;
- c) Os cabos podem ser lançados em distâncias maiores.

As desvantagens de um sistema de comunicação utilizar cabos a Fibra óptica são:

- a) Custo elevado;
- b) Manutenção complicada.

b) Fast Ethernet

O *Fast Ethernet* é 10 vezes mais rápido que o sistema 10Base, surgindo, assim, uma nova geração para a Ethernet, conhecida como padrão IEEE 802.3u. O *Fast Ethernet* é basicamente formado pelas mesmas características do padrão Ethernet, ou seja, possui o mesmo modo de endereçamento, formato do pacote, tamanho e mecanismo de erro. Porém, utiliza novos sistemas de cabeamento.

Há três meios de transmissão no *Fast Ethernet* (100Mbps): 100Base-T4, 100Base-TX e 100Base-FX [20]:

- a) O tipo T4: utiliza quatro canais de par trançado, suportando somente o modo de transmissão *half-duplex*;
- b) O TX: emprega dois cabos de par trançado, suportando transmissões *half-duplex* e *full-duplex*;
- c) O FX: utiliza fibra óptica, com transmissão nos modos *half-duplex* e *full-duplex*.

Na Tabela 3.3 apresentam-se as principais características da geração *Fast Ethernet*.

Tabela 3.1: Cabeamento para a rede Fast Ethernet.

Parâmetros	100Base-T4	100Base-TX	100Base-FX
Tipo de Cabo	Par trançado	Par trançado	Fibra ótica
Topologia física	Estrela	Estrela	Estrela
Conector utilizado	RJ-45	RJ-45	ST
Slot máximo	100	100	2000
Comprimento máximo da rede (metros)	210	210	210
Número de nós por segmentos	1	1	1
Vantagens	Usa cabo UTP	Full-duplex	Full-duplex
Velocidade	100Mbps	100Mbps	100Mbps

c) Gigabit Ethernet

Para atender a aplicações mais complexas como, videoconferências e telemedicina, surge a necessidade de redes locais com maior velocidade. Assim, depois dos padrões 10 e 100 Mbps, o grupo de trabalho 802.3 do IEEE começou a desenvolver, em 1997, o padrão *Gigabit Ethernet* [21].

Este padrão agregou não só ao tráfego de dados, como também de vídeo e voz.

O *Gigabit Ethernet* suporta o quadro do padrão Ethernet mantendo, desta forma, a total compatibilidade com a Ethernet e a *Fast Ethernet*, não sendo necessário à tradução de quadros.



Possui a taxa de transmissão de 1 Gbps e segue o padrão Ethernet na detecção de colisão [21]. Algumas características do *Gigabit Ethernet* são apresentadas na Tabela 3.4.

Tabela 3.1: Cabeamento para o Gigabit Ethernet.

Parâmetro	1000Base LX	1000Base SX	1000Base CX	1000Base T
Tipo de Cabos	Fibra ótica	Fibra ótica	Cabo coaxial	Par trançado
Comprimento Máximo entre dois nós (metros)	550 a 5000	260 a 550		100
Comprimento Máximo do segmento (metros)	550 a 5000	260 a 550	25	100
Número Máximo de nó por segmento	1	1		1
Vantagens.	Usada com backbone em campos	Usada com backbone em campos e edifícios		
Velocidade	1 Gbps	1 Gbps	1 Gbps	1 Gbps

As vantagens da *Gigabit Ethernet* são:

- a) Compatibilidade dos cabeçalhos com outros padrões Ethernet;
- b) Baixo custo, devido à cota de mercado que a Ethernet já tem no mundo;
- c) Evolução natural das redes existentes na empresa que já utilizam a Ethernet;
- d) Simplicidade e facilidade na administração do sistema.

As desvantagens da *Gigabit Ethernet* são:

- a) Limitações no tráfego para multimídia;
- b) Necessidade de encaminhamento de alto nível para ter configurações de rede complexa.



A tendência da tecnologia Ethernet é a consolidação do sistema 10 *Gigabit Ethernet Alliance*. Devido ao crescimento rápido das redes e ao grande volume de tráfego da Internet, tem-se aumentado a demanda da infra-estrutura existente, motivando o desenvolvimento das redes cada vez mais rápidas. Apresentações quantitativas têm sido feitas em relação ao padrão 802.3 evidenciando oportunidades de mercados. Prevê-se que o sistema 10 Gbps 802.3 vai aumentar a capacidade da Ethernet, tornando-se uma excelente tecnologia de banda larga para sistemas distribuídos, processando formação de imagens médicas e diversos aplicativos e derrubando antigos opositores quanto ao determinismo da Ethernet.

Com relação ao surgimento de *Ethernet Alliance*, afirma-se que existem cerca de 140 participantes, representando cerca de 55 empresas, que contribuem para as padronizações do 10 Gbps 802.3. Isto mostra o nível de importância que tal padronização representa para o desenvolvimento da automação, por um grande grupo de fabricantes, garantindo, assim, uma vasta variedade de equipamentos, suportando uma vasta gama de aplicações [22].

Os objetivos do *Ethernet Alliance* são preservar o tamanho e formato do quadro atual do 802.3, operar somente em modo *full duplex*, trabalhar com topologia Estrela nas redes LAN e sendo o meio físico a fibra óptica. O emprego da fibra óptica é para atingir um alcance de até 40 km. A transmissão no modo *full-duplex*, que só é possível com o uso de um *switch*, elimina as colisões de pacotes obtendo-se, assim, ganhos de desempenho consideráveis e garantindo-se que a rede seja determinística para diversas aplicações [23] [24]. Outro avanço da tecnologia Ethernet está no encapsulamento de diferentes protocolos de barramentos de campo, tornando esta rede interoperável, a qual constitui uma característica importante para automação industrial [25].

3.4 Tipos de Comunicação Propostos no Padrão IEEE1451

3.4.1 Cliente/Servidor

O modelo Cliente/Servidor é uma aplicação típica para sistemas distribuídos. Os principais componentes desse modelo são:



- um conjunto de servidores (NCAPs) independentes que oferecem serviços para outros sub-sistemas;
- um conjunto de clientes (supervisor) que solicita a medição e executa atividades de controle baseando-se nas informações fornecidas pelos servidores;
- uma rede de computadores (por exemplo a Ethernet) que permite que os clientes acessem esses serviços.

Os clientes devem saber os “nomes” dos servidores disponíveis e os serviços que os mesmos provêem. Entretanto, servidores não precisam conhecer a identidade dos clientes ou quantos clientes tentam obter seus serviços. Um exemplo de um sistema desenvolvido a partir do modelo cliente-servidor é mostrado na Figura 3.12.

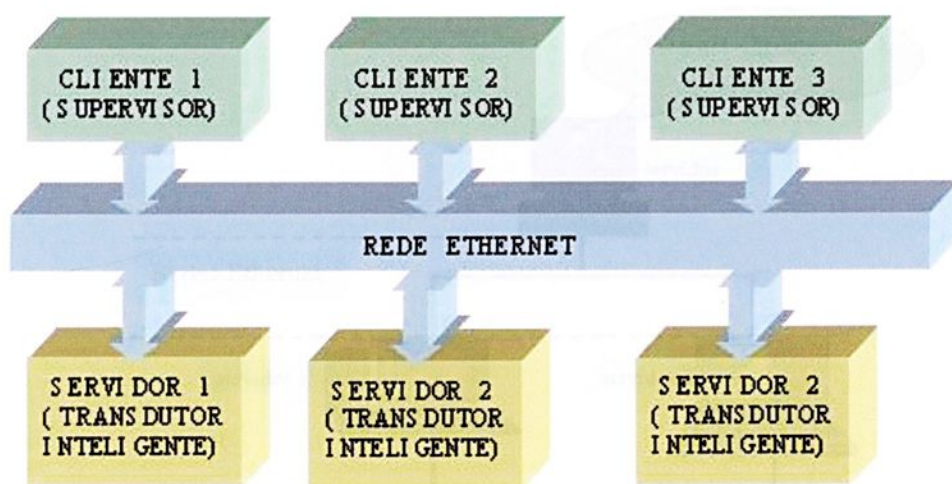


Figura 3.12: Arquitetura de comunicação cliente/servidor.

As vantagens mais importantes do modelo cliente-servidor são: a maneira simplificada de distribuir as informações e a facilidade para ampliar a quantidade de elementos nesta arquitetura de comunicação. Torna-se simples adicionar um novo servidor e integrá-lo gradualmente com o resto do sistema, ou mesmo, atualizar os servidores sem afetar outras partes do sistema. A funcionalidade do NCAP com o STIM insere o transdutor em um modelo cliente/servidor.

3.4.2 Sistemas Distribuídos SD

Os sistemas distribuídos são um retrato da automação atual, sendo uma importante ferramenta para o desenvolvimento do NCAP. Um sistema distribuído pode ser definido como uma integração de sistemas computacionais autônomos, combinados de forma a atingir um objetivo comum, sendo que os sistemas computacionais envolvem os diversos tipos hardware e software. No diagrama mostrado na Figura 3.13 o servidor recebe a requisição de serviço via Internet. O servidor deve passar a solicitação de serviço ao *servidor 1* ou ao *servidor 2*, conforme o serviço requisitado. Cada servidor controla diferentes tipos de dispositivos, com diferentes softwares de gerenciamento. A rede de controle ou barramento de campo é a rede Ethernet. O esquema apresentado na Figura 3.13 é um típico sistema distribuído.

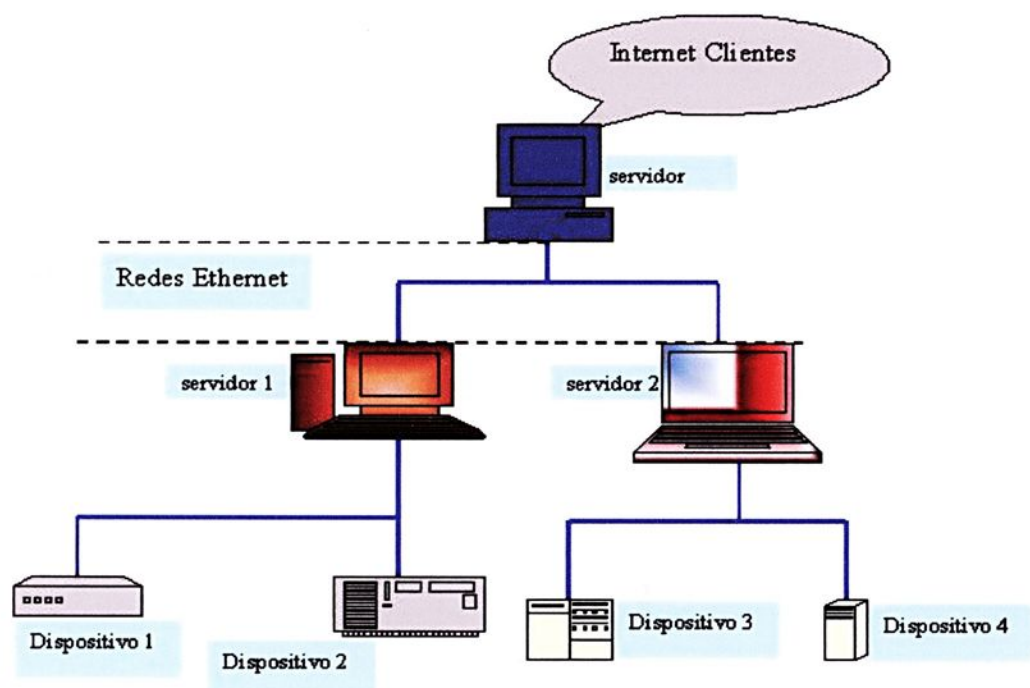


Figura 3.13: Configuração de um sistemas distribuído.

As vantagens adquiridas com o emprego de um sistema distribuído para desenvolver um transdutor inteligente baseado no padrão IEEE1451 são:

- a) Compartilhamento de recursos: permite que mais de um usuário tenha acesso aos serviços de um transdutor;
- b) Flexibilidade: se um NCAP falhar, os demais continuam em funcionamento;
- c) Escalabilidade: o número de transdutores inteligentes é estabelecido de acordo com a aplicação;
- d) Desempenho: permite a reconfiguração de um transdutor conectado na rede, sem inutilizar a rede.

No próximo capítulo, descreve-se os conceitos básicos da tecnologia Java, que foi utilizada para implementar a parte lógica do NCAP.



4. Tecnologia Java

4.1 Introdução ao Java

Tudo começou quando o engenheiro *Patrick Naughton* decidiu sair da Sun Microsystems, alegando que a empresa estava no caminho errado no desenvolvimento de tecnologia. O presidente da Sun, na época, que era *Scott McNealy*, amigo pessoal de *Naughton*, pediu um relatório informando tudo que ele sentia que a Sun estava fazendo de errado, com sugestões de como eliminar esses problemas. *Naughton* fez o relatório e *McNealy* o passou para todos os executivos da Sun. Com o efeito do relatório, a Sun decidiu propor a *Naughton* todo o subsídio necessário para desenvolver um projeto na empresa. *Patrick Naughton* aceitou o desafio e fez algumas exigências: o hardware e o software usado no projeto não poderiam ser compatíveis com a arquitetura Sun, solicitou um orçamento de 1 milhão de dólares para o primeiro ano e que o projeto seria elaborado fora das dependências da Sun. A Sun aceitou a sua solicitação e montou uma equipe que foi liderada por *Naughton*, pelo também engenheiro *Mike Sheridan* e pelo programador *Jim Gosling*. A equipe foi denominada de *Green* e foi composta por um total de treze pessoas [26].

No princípio, a equipe *Green* pretendia desenvolver um dispositivo eletrônico inteligente para o consumidor final, com capacidade de se comunicar de diversas formas e ampliando suas funcionalidades. O responsável por desenvolver a arquitetura desse dispositivo foi o engenheiro de hardware Ed Frank. Porém, depois de meses de trabalho, a equipe decidiu criar uma nova plataforma para desenvolver equipamentos e construir um software com capacidade de ser suportado pelos mais diferentes dispositivos. O ponto forte da equipe *Green* estava na programação, pois *Naughton*, apesar da formação de engenheiro, desenvolvia projetos como programador. Outro líder, *Gosling*, escreveu sua primeira linguagem de programação com 14 anos e era considerado um dos melhores programadores de sua época. Então, resolveram criar



uma linguagem de programação que pudesse conter tudo o que a equipe considerava importante. Surgiu, então, uma linguagem baseada em C e C++ que denominaram *Oak* (carvalho) em homenagem a uma árvore que dava para a janela do escritório de *James Gosling*. Porém, descobriu-se, mais tarde, que já havia uma linguagem de computador chamada *Oak*. Então, a equipe resolveu mudar o nome para Java, inspirados pelo nome da cidade de onde provinha o café que costumavam tomar quando se reuniam para bate-papo. Contudo, o mercado de dispositivos inteligentes pessoais não estava se desenvolvendo como a equipe imaginava, e o projeto *Green* começou a sofrer algumas dificuldades [27].

Para sobreviver, a Sun apostava em um grande contrato que estava concorrendo, mas foi vencida pela concorrência. Quando o projeto já estava sendo considerado um fracasso, veio a explosão da internet, mais precisamente, a *World Wide Web* (www), e então, a Sun viu a grande oportunidade de utilizar Java para criar páginas da *Web*, dando-se nova vida ao projeto. Em meados de 1995 a Sun anunciou a linguagem de programação Java em uma conferência. O evento, de pouca expressão, gerou interesse imediato na comunidade comercial devido à grande expansão da www. Então, a explosão de popularidade da linguagem Java coincide com a explosão da Internet e, desta forma, muitas pessoas associam a linguagem Java à criação de páginas da internet ou à execução de aplicativos para Internet. Entretanto, Java atualmente é aplicado em diversas áreas da engenharia, tais como:

- a) Desenvolvimento de microprocessadores;
- b) Tecnologia embarcada;
- c) TV Digital;
- d) Aplicativos para celulares;
- e) Projetos da NASA, como o robô “*Spirit*”²⁰;
- f) Controle de dispositivos eletrônicos.

A tecnologia Java é uma linguagem de programação orientada a objeto que suporta alto nível de abstração, possui um conjunto de Interfaces de Programas Aplicativos (*Applications Programming Interface APIs*) e pode executar suas aplicações em diferentes tipos de

²⁰ Spirit: o robô apelidado de Spirit foi projetado para explorar a superfície do planeta Marte.



dispositivos. O ambiente de execução foi denominado, pela equipe *Green*, de Máquina Virtual Java (*Java Virtual Machine* ou *JVM*) [28].

4.2 Peculiaridades da linguagem de programação Java

A tecnologia Java tem algumas peculiaridades que merecem destaque: a JVM e o Coletor de Lixo (*Garbage Collector*).

4.2.1 A JVM

A Sun define a Máquina Virtual Java como sendo uma máquina imaginária que é implementada via software e hardware. Uma máquina virtual é um programa de computador que simula o funcionamento de um computador virtual, dentro da memória de um computador real. Desta forma, é possível executar um programa Java em qualquer ambiente que suporte a Máquina Virtual Java. Além disso, a máquina virtual Java isola o programa do restante do microprocessador.

Quando um código fonte escrito em Java é traduzido pelo compilador (*javac*), gera os chamados *bytecode* (códigos de bytes), isto é, o código de máquina de um processador virtual. A JVM é um programa capaz de interpretar os códigos criados pelo compilador, ou seja, uma vez compilado um programa Java, este pode ser executado em qualquer plataforma que esteja dotada da JVM. Daí surge o slogan da Sun para linguagem Java: “*write once run anywhere*” (escreva uma vez e rode em qualquer lugar).

Um código para ser interpretado pela JVM deve ter assinatura (**.class**) e possuir um código compatível com as instruções Java. Assim, a máquina virtual Java traduz as instruções do código Java, para instruções válidas para a plataforma na qual está instalada (Windows, Solaris, Linux, celular etc). Na Figura 4.1 apresenta-se a arquitetura funcional da linguagem Java, sendo que o programa denominado de *ProgramaJava*, o qual possui a extensão **.java** (*ProgramaJava.java*), é compilado. Nesta etapa, o compilador gera os *bytecodes* que são



armazenados em um arquivo **.class** (*ProgramaJava.class*). Este é o arquivo que é interpretado pela Máquina Virtual Java para códigos de máquinas e onde sua portabilidade está adquirida, pois, agora, o *ProgramaJava* pode rodar em qualquer máquina que possui a JVM.

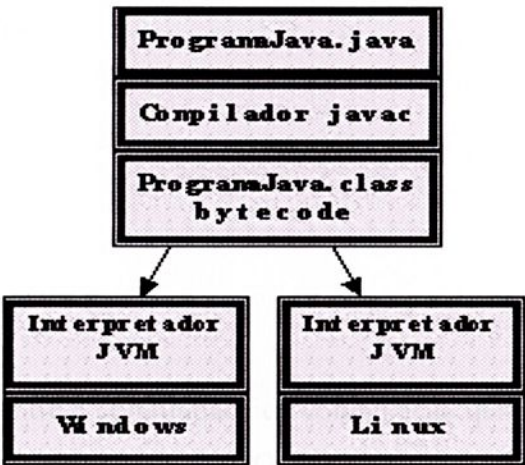


Figura 4.1: A arquitetura da tecnologia Java.

4.2.2 Coletor de Lixo

Em Java o programador não precisa se preocupar em desalocar memória, pois, quando os programas são executados, há uma verificação das variáveis que estão sendo utilizadas, e desloca-se automaticamente as variáveis com referências iguais a zero.

4.3 Características da linguagem de programação Java

A linguagem Java é considerada:

- a) orientada a objeto (OO);



- b) distribuída;
- c) segura;
- d) robusta;
- e) independente da arquitetura;
- f) interpretada.

4.3.1 Orientada a Objeto

Este tema causa divergências entre a comunidade da computação quanto às características da OO, porém, uma definição muito comum para OO é: “*Orientado a objeto: dirigido para qualquer coisa em que se possa pensar*” [16].

Assim, trata-se de uma linguagem puramente orientada a objeto, pois possui a sintaxe de programação baseada em objetos e classes, oferecendo mecanismo de encapsulamento, polimorfismo e hereditariedade, características fundamentais de uma linguagem OO.

a) Classe

Uma classe é a matriz a partir do qual os objetos são criados (instanciados). Os objetos possuem a mesma estrutura e definem o comportamento da classe da qual eles foram instanciados. A distinção entre classe e objeto é que, o objeto equivale a uma variável em programação convencional, pois especifica uma área de armazenamento, enquanto que a classe é vista como um tipo abstrato de dados²¹, uma vez que representa um molde a ser seguido pelos objetos. Assim, uma classe pode possuir múltiplas instâncias (objetos) que compartilham as mesmas operações, porém, com valores diferentes.

²¹ Tipo Abstrato de Dados: fornece abstração sobre uma estrutura de dados (classes) em termos de uma operação bem definida



b) Encapsulamento

O encapsulamento é o ato de ocultar detalhes de um programa, que não desejam ser revelados. Em Java pode-se estabelecer o controle de acesso a métodos²², através das palavras reservadas: *private*, *protected* e *public*.

- a) *private*: os métodos são acessíveis somente pelos objetos da própria classe;
- b) *protected*: os métodos são acessíveis pelos objetos da classe e subclasse;
- c) *public*: os métodos podem ser acessados por todos os objetos de todas as classes.

c) Polimorfismo

O polimorfismo é a habilidade de assumir várias formas. Em Java o polimorfismo pode ocorrer quando:

- a) um único método pode ser definido em mais de uma classe e pode assumir diferentes implementações em cada uma dessas classes;
- b) uma variável pode apontar para objetos de diferentes classes em instantes diferentes.

d) Hereditariedade

A herança em Java consiste na reutilização de métodos pertencentes a uma outra classe, ou seja, pode-se criar uma subclasse utilizando os recursos de uma outra superclasse já construída, ou modificando alguns recursos desta classe. Java permite somente a herança simples, isto é, uma subclasse herda os recursos de uma única superclasse. A herança é definida através da palavra-chave *extends*.

²² Métodos em Java: são responsáveis por executar uma tarefa específica dentro de uma classe.



4.3.2 Distribuída

Java possui ferramentas que facilitam o compartilhamento de informações e trabalho conjunto. A API Invocação de Métodos Remotos (*Remote Method Invocation* ou RMI) utiliza o modelo de comunicação cliente/servidor para implementar métodos remotos e, o protocolo TCP/IP²³, para transportar os métodos remotos. Junto com a capacidade de executar programas Java e diferentes plataformas, a tecnologia Java torna-se muito atraente para desenvolver aplicações distribuídas.

4.3.3 Segura

A política de segurança em Java é implementada com base no conceito de *sandbox*. No sistema *sandbox* há um controle no limite referente ao acesso dos recursos do meio, no qual o programa deve ser executado. Alguns componentes que compõe o *sandbox* Java são: segurança na programação Java, o verificador de *bytecode* (*bytecode verifier*), o *class loader*, o *security manager* e o *access controller*.

- a) Segurança na programação Java: ao desenvolver um programa em Java, o programador deve ficar atento para os níveis de acesso para os objetos e os elementos primitivos de dados. Os níveis de acesso são: default, public, protected e private;
- b) O verificador de *bytecode*: tem como objetivo verificar se as classes Java contidas nos *bytecode* seguem as estruturas de programação Java. Este recurso reforça a proteção da memória de programas Java;
- c) O *class loader* é o responsável por carregar as classes Java na máquina Virtual. O *class loader* é o único que sabe de onde as classes são originadas e se ela foi assinada ou não;

²³ TCP/IP Transmission Control Protocol/Internet Protocol arquitetura de comunicação com endereço definido na rede



Assim, se as classes são encontradas pelo *class loader*, a operação continua, caso contrário, a exceção *ClassNotFoundException* é gerada. O *class loader* interage com *security manager* e *access controller* para determinar a política de segurança de um programa. Por essas características o *class loader* é fundamental para a política de segurança Java;

- d) O *security manager*: é a componente que define em última instância se um programa pode ou não ser executado. Geralmente é utilizado em aplicações remotas;
- e) O *access controller*: é responsável em controlar a permissão de acesso aos recursos. Como a JVM está inserida na memória do computador, a segurança da máquina virtual depende também da robustez da memória, e assim, a utilização de memórias do tipo *Error-Conection Code* (ECC) corrige erros ocasionados na memória, podendo evitar ataques a JVM.

A política de segurança Java suporta a criptografia de chave pública da RSA²⁴, o que possibilita um maior proteção de privacidade.

4.3.4 Robusta

Quanto mais robusto um software, mais confiável ele o será. A robustez da linguagem Java é adquirida através das seguintes características:

- a) não utiliza ponteiros, evitando uma série de erros relacionados com esse tipo de estrutura;
- b) a coleta de lixo faz a desalocação automática de memória, evitando objetos ociosos;
- c) possui o tratamento de erros em tempo de execução;
- d) exige que objetos e números sejam definidos durante a compilação. O compilador evita passar qualquer indefinição em relação ao tipo de dados. Essa característica garante uma maior segurança do código e torna menos sujeito a erros.

²⁴ A sigla RSA é uma referência que corresponde as iniciais dos inventores do código, R. L. Rivest, A. Shamir e A. Adleman que na época trabalhavam no Massachussets Institute of Tecnology (M.T.I).



4.4 Instalação de um Kit de Desenvolvimento Java.

A seguir, são descritas as etapas para a instalação JDK:

- 1) no site *http://java.sun.com/j2se/*, escolha a versão que se deseja trabalhar, por exemplo, a versão 1.4.2;
- 2) clique para realizar o download de acordo com o sistema operacional;
- 3) aceite a licença para utilizar o kit;
- 4) copie o JDK no diretório adequado, por exemplo, *c:\j2sdk1.4.2*;
- 5) crie o caminho como, por exemplo: *SET PATH=;c:\j2sdk1.4.0\bin* e *SET CLASSPATH=.;c:\j2sdk1.4.0\jre\lib* (Windows).

Após a instalação, o kit disponibiliza as seguintes pastas:

- a) *bin*: contém os programas necessários para compilação e execução como, por exemplo, o *javac* e o *rmic*;
- b) *jre*: o *Java Run-Time Environment* que contém os programas e bibliotecas necessários para executar o código Java;
- c) *jre\bin*: contém os arquivos necessários para rodar programas Java, de acordo com o sistema operacional;
- d) *jre\lib*: contém as bibliotecas básicas para a execução e compilação de programas Java;
- e) *demo*: contém vários programas em Java, para servir como exemplo;
- f) *include*: contém interfaces para programas em Java e C;
- g) *lib*: contém arquivos executáveis.

No Apêndice A, apresentam-se alguns sites úteis para assuntos relacionados com a tecnologia Java.



4.4.1 Pacotes da linguagem Java

A linguagem Java possui muitas classes predefinidas que são agrupadas por categorias denominadas de pacotes. Esses pacotes são referidos como Java API [29]. Na Tabela 4.1 mostram-se alguns pacotes Java.

Tabela 4.1: Pacotes pertencentes à linguagem de programação Java.

Pacote	Descrição
java.applet	contém várias classes que permitem a criação de <i>applets</i> .
java.awt	o pacote contém as classes e interfaces exigidas para criar e manipular interfaces gráficas.
java.io	suporta entrada e saída de dados.
java.lang	possui classes e interfaces requeridas por muitos programas Java. Este pacote é importado automaticamente
java.net	permite aos programas comunicar-se via rede.
java.rmi	contém classes e interfaces que permitem ao programador, chamar métodos pertencentes a JVM diferentes.
java.util	pacote responsável por definir classes de utilitários e interfaces para manipular datas, horas, armazenamento de grande quantidade de dados, etc.
java.swing	possui classes e interfaces para componentes GUI
javax.comm	Pacote que contém classe e interfaces que permitem a comunicação com a porta serial e porta paralela.

Fonte: Deitel, H. M. e Deitel P. J. Java como Programar, Bookman.

A tecnologia Java possui três plataformas distintas: J2ME, J2SE e J2EE. Cada plataforma tem áreas específicas de atuações:

- a) J2ME - Java 2 Micro Edition, utilizado em dispositivos móveis;
- b) J2SE - Java 2 Standard Edition, voltado para aplicações em estação de trabalho;
- c) J2EE - Java 2 Enterprise Edition, empregado em atividades para servidores.



Os pacotes podem ser acessados em um programa através da palavra reservada *import*, como ilustra-se na Figura 4.2.

```
import java.awt;  
import javax.swing;  
import java.io;  
import java.rmi;  
import javax.swing.JOptionPane;
```

Figura 4.1: Utilizando o comando *import* em Java.

4.5 Sintaxes da programação Java

Na Tabela 4.2 mostram-se as palavras-chaves que não podem ser usadas como identificadores ou variáveis em Java. Essas palavras são reservadas pela linguagem para implementar vários recursos.

Tabela 4.1: Palavras-chaves.

Abstract	Boolean	break	byte	case
Catch	char	class	continue	default
Do	double	else	Extends	false
Final	finally	float	For	if
Implements	import	instanceof	int	interface
Long	native	new	null	package
Private	protected	public	return	short
Static	super	switch	synchronized	this
Throw	throws	transient	true	try
Void	volatile	while		

Fonte: Deitel, H. M. e Deitel P. J. Java como Programar, Bookman.



Na Tabela 4.3 lista-se os tipos de dados primitivos em Java. Os tipos primitivos são bases para a construção de tipos mais complicados. Todas as variáveis devem possuir um tipo definido antes de serem usadas em um programa Java.

Tabela 4.2: Tipos de dados primitivos em Java.

Tipo	Tamanho em bits
Byte	8
Short	16
Int	32
Long	64
Float	32
Double	64

Fonte: Deitel, H. M. e Deitel P. J. Java como Programar, Bookman

A linguagem Java suporta vários operadores destinados a realização de operações aritméticas, lógicas, de comparação e de atribuição. Na Tabela 4.4 apresenta-se o significado e citam-se exemplos dos operadores aritméticos. As regras de precedências são as mesmas utilizadas em álgebra, ou seja, os operadores em expressões dentro de pares de parênteses são avaliados primeiramente, as operações de multiplicação, divisão e de módulo são aplicadas em seguida e as operações de adição e subtração são aplicadas por último.

Tabela 4.1: Operadores Aritméticos.

Operadores	Significado	Exemplo
+	Soma	x + y
-	Subtração	x - y
*	Multiplicação	x * y
%	Resto da divisão inteira	x % y
/	Divisão	x / y
+	Sinal positivo(+unitário)	+ x
-	Sinal negativo(-unitário)	- y
++	Incremento unitário	++ x
--	Decremento unitário	-- x

Fonte: Deitel, H. M. e Deitel P. J. Java como Programar, Bookman.



Na Tabela 4.5 apresenta-se o funcionamento dos operadores lógicos.

Tabela 4.2: Operadores Lógicos.

Operadores	Significados
&&	Operação lógica E (AND)
	Operação lógica OU (OR)
!	Negação lógica
&	Comparação bit-a-bit E (AND)
	Comparação bit-a-bit OU (OR)
^	Comparação bit-a-bit OU-Exclusivo (XOR)
<<	Deslocamento à esquerda
>>	Deslocamento à direita
>>>	Deslocamento à direita com preenchimento de zeros
-	Complemento bit-a-bit
x <<= y	Atribuição com deslocamento a esquerda (x = x << y)
x >>= y	Atribuição com deslocamento a direita (x = x >> y)
x >>>= y	Atribuição com deslocamento a direita e com preenchimento de zeros (x = x >>> y)
x &= y	Atribuição AND (x = x & y)
x = y	Atribuição OR (x = x y)
x ^= y	Atribuição XOR (x = x ^ y)

Fonte: Deitel, H. M. e Deitel P. J. Java como Programar, Bookman.

Na Tabela 4.6 descreve-se o funcionamento dos operadores de comparação na linguagem de programação Java.

Tabela 4.3: Operadores de Comparação.

Operador	Significado	Exemplo
= =	Igual	x = = y
!=	Diferente	x != y
<	Menor que	x < y
>	Maior que	x > y
<=	Menor ou igual	x <= y
>=	Maior ou igual	x >= y

Fonte: Deitel, H. M. e Deitel P. J. Java como Programar.

Na Tabela 4.7 são apresentados os operadores de atribuição e como eles podem ser utilizados.



Tabela 4.1: Operadores de Atribuição.

Expressão	Significado
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$

Fonte: Deitel, H. M. e Deitel P. J. Java como Programar, Bookman.

Na Tabela 4.8 descreve-se o funcionamento das seqüências de escape utilizadas em Java.

Tabela 4.2: Seqüência de escape.

Seqüência de Escape	Descrição
<code>\n</code>	Nova linha. Posiciona o cursor da tela no início da próxima linha.
<code>\t</code>	Tabulação Horizontal. Mostra o cursor da tela para próxima parada de tabulação.
<code>\r</code>	Retorno de Carro. Posiciona o cursor de tela no início da linha atual; não avança para a próxima linha.
<code>\\</code>	Barra invertida. Utilizada para imprimir um caractere barra invertida.
<code>\"</code>	Aspas duplas. Utilizada para imprimir um caractere aspas duplas.

Fonte: Deitel, H. M. e Deitel P. J. Java como Programar, Bookman.

Na Tabela 4.9 resumem-se as funções de incremento e decremento de um operador. Se um operador de incremento ou decremento é inserido antes de uma variável, esta será incrementada ou decrementada, respectivamente.

Tabela 4.3: Operadores de incremento e decremento.



Operador	Expressão	Comentário
++	++variável	incrementa a variável por 1, e então utiliza o novo valor da variável no evento.
++	variável++	utiliza o valor da variável no evento, e então incrementa a variável por 1.
--	-- variável	decrementa a variável por 1, e então utiliza o novo valor da variável no evento.
--	Variável--	utiliza o valor da variável no evento, e então decrementa a variável por 1.

Para inserir comentários dentro de um programa Java, utiliza-se:

- a) // comentário de uma linha;
- b) /* Comentário de várias linhas */;
- c) /** Comentário de documentação */.

4.6 Estruturas de seleção em Java

4.6.1 Estrutura de seleção if

A palavra reservada *if* pode ser empregada da seguinte forma:

if (se) ocorrer um evento desejado, então,
execute uma ação

Como exemplo, tem-se:

if (x > 10)


```
System.out.println(" Imprime o valor de x" + x)
```

4.6.2 Estrutura de seleção *if/else*

As palavras reservadas *if/else* são empregadas como:

if (se) ocorrer um evento desejado, então,
execute uma ação

else (senão)
execute uma outra ação.

Como exemplo, cita-se:

```
if (x > 10)
```

```
System.out.println(" Imprime o valor de x" + x);
```

```
else
```

```
System.out.println(" Imprime o valor de y" + y)
```

4.6.3 Estrutura de repetição *while*

A palavra reservada *while* é empregada como:

while (enquanto) ocorrer um evento
execute uma ação

Como exemplo, considera-se:



while (x <= 100)

 x = 2 * x;

4.6.4 Estrutura de repetição *for*

A palavra reservada *for* é utilizada na forma:

for (expressão1; expressão2; expressão3)

 Execute uma ação

Como exemplo, tem-se:

for (int x = 2; x <= 100; x++)

 contador += x;

4.6.5 Estrutura seleção múltipla *switch*

A estrutura *switch* consiste de uma série de eventos diferentes e um caso *default* opcional:

switch (evento)

case 1:

 executa o evento relacionado ao caso 1

break;

case 2:

 executa o evento relacionado ao caso 2

break;

case 3:

 executa o evento relacionado ao caso 3



break;

case 4:

executa o evento relacionado ao caso 4

break;

default:

4.6.6 Estrutura de repetição *do/while*

As palavras reservadas *do/while* são usadas na forma:

do {

evento

} while (condição);

Como exemplo, ilustrativo, cita-se:

do {

$x = 2 * x...$

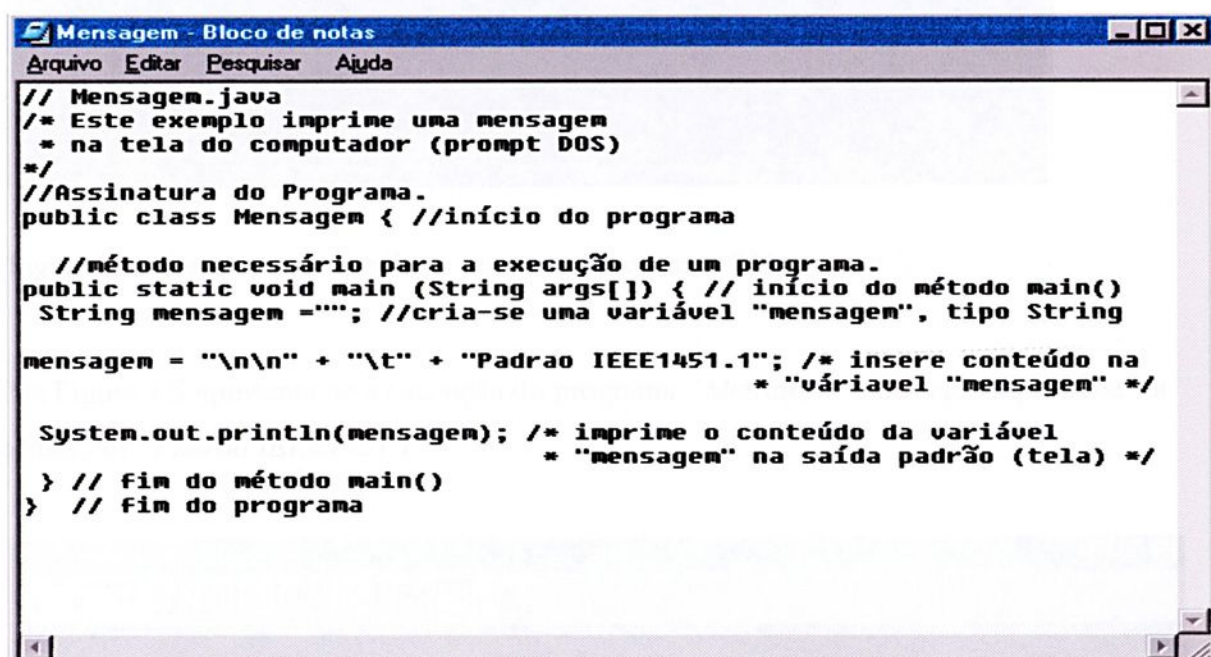
} while ($x \leq 100$);

4.7 Exemplo de programação Java

Com o objetivo de ilustrar o comportamento de alguns programas Java, implementaram-se dois exemplos. Na Figura 4.3 apresenta-se o código fonte do programa denominado de “*Mensagem*”. Este programa deve imprimir na tela do computador a mensagem “Padrão IEEE1451.1”. O programa utiliza o bloco de notas da plataforma *Windows* para editar o código, porém, qualquer editor de texto pode ser utilizado. Para desenvolver este programa, cria-se uma



variável do tipo *String* (denominada no programa de *mensagem*), insere o texto (Padrão IEEE1451.1) na variável e através do método *System.out.println()*, imprime na tela do computador, o conteúdo da variável *mensagem*.



```
// Mensagem.java
/* Este exemplo imprime uma mensagem
 * na tela do computador (prompt DOS)
 */
//Assinatura do Programa.
public class Mensagem { //início do programa

    //método necessário para a execução de um programa.
    public static void main (String args[]) { // início do método main()
        String mensagem =""; //cria-se uma variável "mensagem", tipo String

        mensagem = "\n\n" + "\t" + "Padrao IEEE1451.1"; /* insere conteúdo na
                                                         * "uáriavel "mensagem" */

        System.out.println(mensagem); /* imprime o conteúdo da variável
                                         * "mensagem" na saída padrão (tela) */
    } // fim do método main()
} // fim do programa
```

Figura 4.1: Exemplo de programação Java.

Na Figura 4.4 ilustram-se os comandos para compilação e a etapa que possibilita a execução do programa “Mensagem”. Ao se digitar *javac Mensagem.java*, tecla-se [Enter]. Ao digitar *java Mensagem* também deve-se teclar [Enter] para executar o programa.

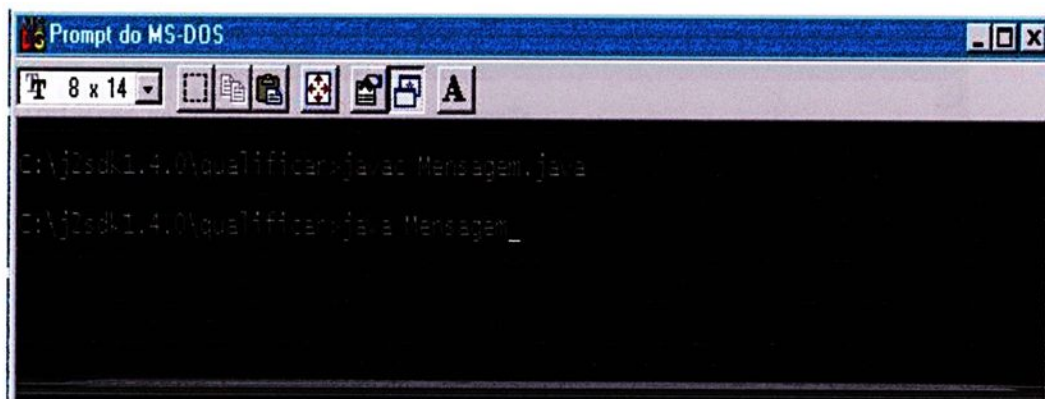


Figura 4.2: Etapas para compilar e executar o programa “*Mensagem*”.

Na Figura 4.5 apresenta-se a execução do programa “*Mensagem*”, ou seja, imprime-se na tela a mensagem “*Padrão IEEE1451.1*”



Figura 4.3: Execução do programa “*Mensagem*”.

Na Figura 4.6 descreve-se o código do programa “*Contagem*”. Neste programa é implementada uma contagem de 1 a 20, sendo que o resultado é mostrado através de uma interface gráfica.

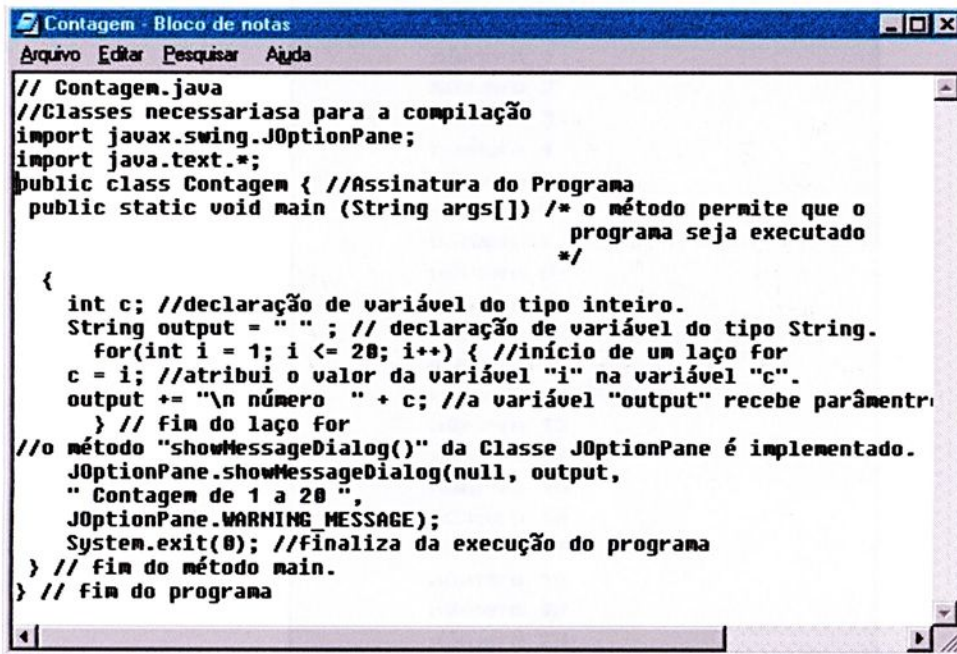


Figura 4.4: Descrição do programa “Contagem”.

Na Figura 4.7 é apresentada a etapa de compilação e a etapa para a execução do programa “Contagem”.

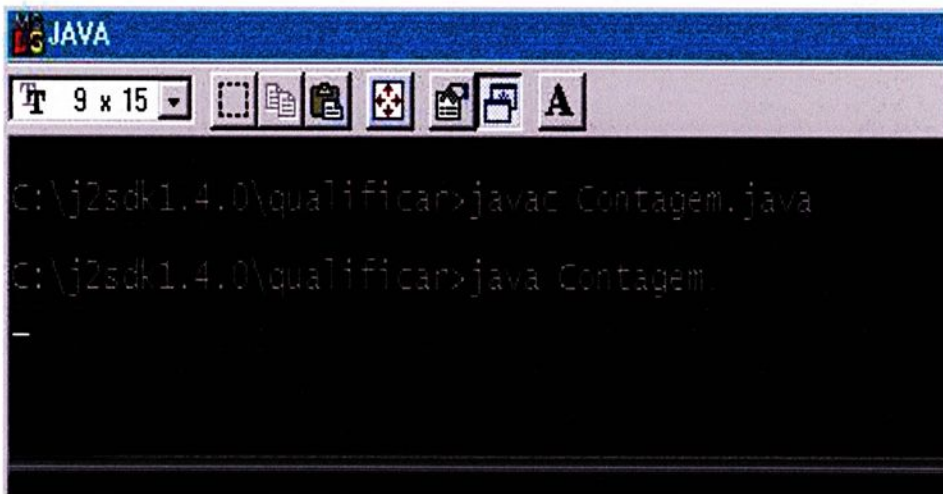


Figura 4.5: Etapas necessárias para compilar e executar o programa “Contagem”.

Na Figura 4.8 mostra-se o resultado da execução do programa “Contagem”.

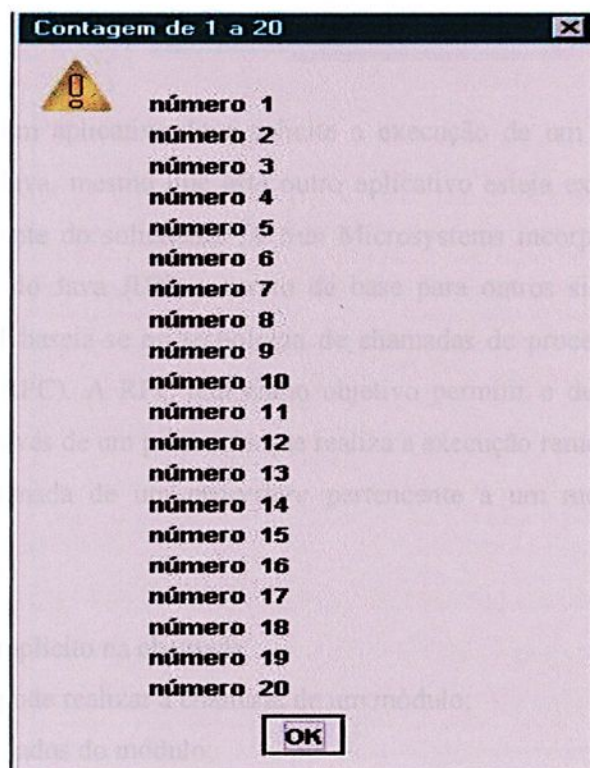


Figura 4.8: Resultado da execução do programa “Contagem”.

Recomenda-se ao leitor, que consulte o Apêndice B na qual apresentam-se outros exemplos de programação Java.

4.8 Java RMI

4.8.1 Introdução

A Invocação de Métodos Remotos (*Remote Method Invocation* ou RMI) é um mecanismo para desenvolver programas, com características funcionais distribuídas [30]. A RMI é usada para desenvolver o modelo de comunicação da parte lógica do NCAP.

O RMI permite que um aplicativo Java solicite a execução de um ou mais métodos remotos de outro aplicativo Java, mesmo que este outro aplicativo esteja executando em uma máquina virtual (JVM) diferente do solicitante. A Sun Microsystems incorporou a tecnologia RMI, a partir da versão 1.1 do Java JDK, e serviu de base para outros sistemas de objetos distribuídos em Java. O RMI baseia-se na tecnologia de chamadas de procedimentos remotos (*Remote Procedure Call* ou RPC). A RPC tem como objetivo permitir o desenvolvimento de aplicações cliente/servidor através de um protocolo que realiza a execução remota de *procedure*.

O RPC realiza a chamada de um *procedure* pertencente a um módulo/processo da seguinte forma:

- a) o módulo/processo é implícito na chamada;
- b) apenas um *procedure* pode realizar a chamada de um módulo;
- c) o *procedure* utiliza os dados do módulo;

O RMI realiza a chamada de método da seguinte forma:

- a) o objeto é explícito na chamada;
- b) vários objetos podem chamar diversos métodos remotos;
- c) o objeto pode ter dados próprios.

4.8.2 Objetivos Java RMI

A arquitetura RMI deve permitir a construção de objetos Java, que sejam executados em uma mesma máquina virtual ou em máquinas virtuais diferentes, e assim, construir um aplicativo distribuído, totalmente escrito em Java.

4.8.3 Vantagens e desvantagens da Java RMI



Como vantagens da tecnologia RMI citam-se:

- a) Não é necessário criar um protocolo para viabilizar o transporte de dados;
- b) É possível usar o Security Manager para evitar distribuições ilegais;
- c) Não é necessário preocupar-se com a troca de objetos entre servidor e cliente;
- d) A RMI permite o transporte de tipos de dados complexos.
- e) O RMI transporta os objetos utilizando o protocolo TCP/IP, o que facilita utilização via Internet.

Constitui desvantagem da tecnologia RMI o fato que só funciona para aplicativos Java.

A implementação de Java RMI combina com a arquitetura cliente-servidor distribuída, sendo construída a partir de três camadas de abstração: a primeira camada refere-se a *skeleton* e o *stub*, a segunda refere-se à camada de referência remota e a terceira refere-se à camada de transporte.

4.8.4 Camada *Skeleton* e *Stub*

Esta camada é visível ao programador e é responsável por possibilitar as chamadas dos métodos. A *skeleton* é uma classe gerada automaticamente na compilação do servidor²⁵ para o uso de RMI. Ela sabe como se comunicar com a *stub* através da camada de referência remota. A *skeleton* estabelece uma conversação com a *stub*. Quando a camada *skeleton* recebe uma chamada, executa os seguintes passos:

- 1) Lê os parâmetros enviados pela *stub*;
- 2) Invoca os métodos no objeto remoto;
- 3) Escreve e transmite o resultado ao objeto que executou a chamada remota.

²⁵ Servidor: neste caso o servidor é o programa responsável por implementar a interface remota.



A camada *stub* é uma classe gerada pelo cliente²⁶. Quando o cliente invoca um método pertencente à outra JVM, a camada *stub* fica responsável por realizar a chamada deste método via objeto remoto.

Quando o *stub* é invocado, executa os seguintes passos:

- 1) Inicia a conexão com a JVM, que contém o objeto remoto;
- 2) Escreve e transmite os parâmetros para a JVM remota;
- 3) Aguarda pelos resultados da invocação do método;
- 4) Lê os resultados retornados;
- 5) Retorna os valores ao objeto que executou a chamada.

4.8.5 Camada de referência remota

A camada de referência remota é responsável por associar os objetos remotos através da sintaxe de programação.

4.8.6 Camada de Transporte

A camada de transporte utiliza o protocolo TCP/IP para transportar os objetos remotos.

Na Figura 4.9 apresenta-se a arquitetura de funcionamento da comunicação cliente-servidor baseada na tecnologia Java RMI. A camada de transporte de uma aplicação RMI, utiliza o protocolo TCP/IP para enviar os objetos remotos.

²⁶ Cliente: neste caso, cliente é o programa responsável por acessar os serviços de um servidor.



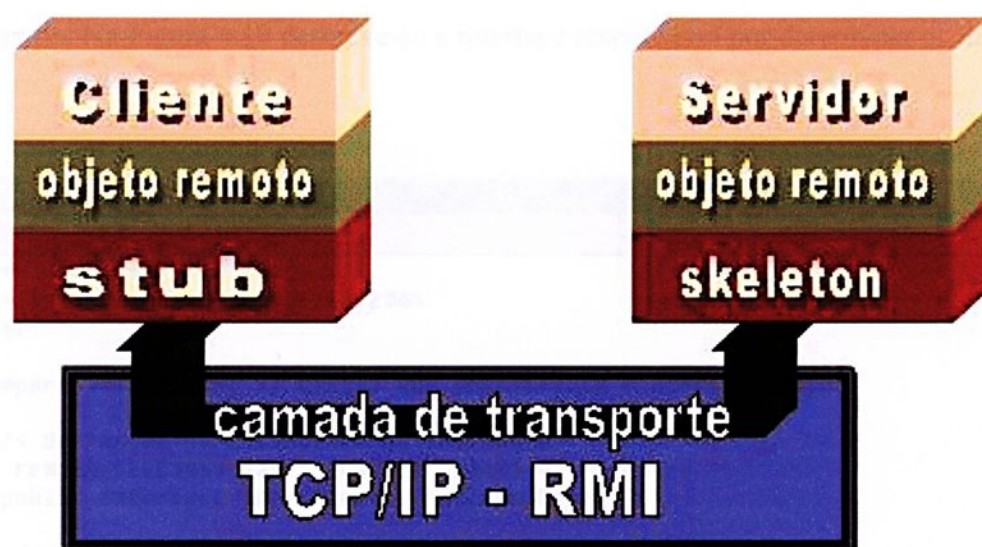


Figura 4.9: Arquitetura RMI

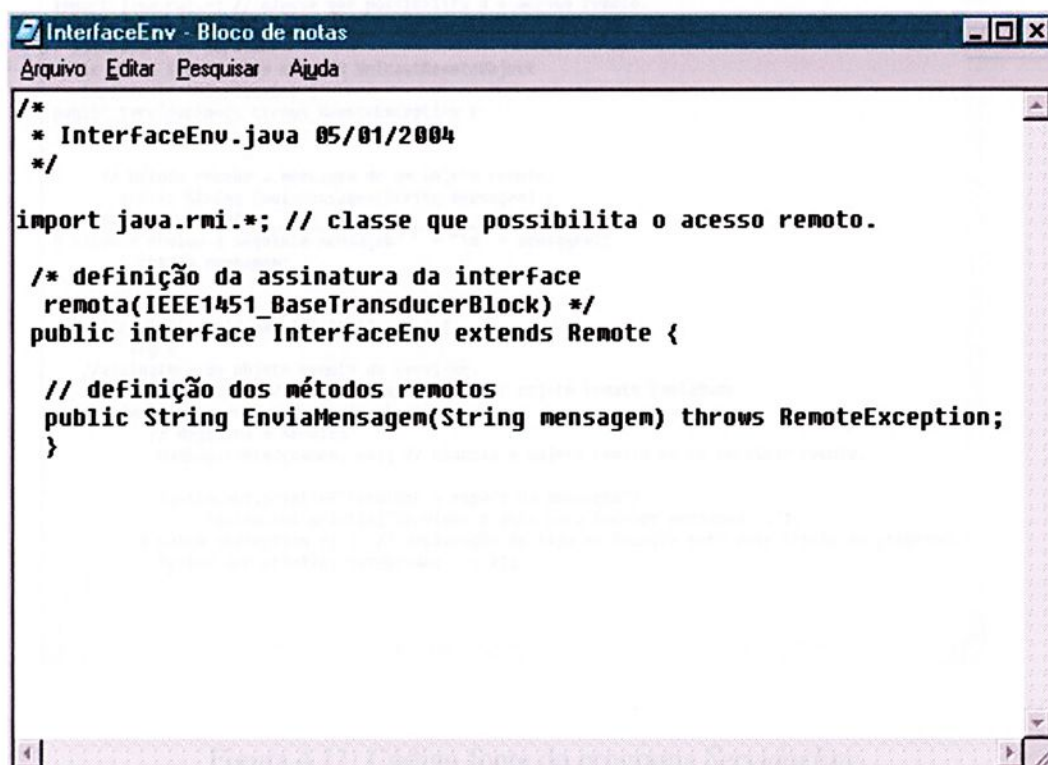
Os aplicativos RMI interagem entre si através de herança e implementações. A interface remota e a classe servidora, herdam os métodos RMI de duas classes disponíveis no pacote *java.rmi*: a classe *Remote* e a classe *UnicastRemoteObject*, respectivamente.

Um aplicativo RMI deve implementar a estrutura do cliente, da interface e do servidor [30]. A interface herda o comportamento da classe *Remote* e declara os métodos que serão invocados remotamente. O servidor herda a classe *UnicastRemoteObject*, implementa o comportamento da interface e deve publicar o método com *Naming.rebind()*. O cliente chama o método remoto ao instanciá-lo através da classe *Naming.lookup()*. Deve ser observado que os métodos *rebind* e *lookup* pertencem a classe *Naming*.

4.8.7 Exemplo de uma aplicação RMI

No exemplo a seguir um cliente deve enviar uma mensagem para o servidor. A aplicação possui três programas. O programa *InterfaceEnv* determina um método remoto denominado de *EnviaMensagem()*. Este método recebe um dado do tipo *String* e deve declarar a exceção

RemoteException. Na Figura 4.10 descreve-se a interface responsável por determinar os métodos remotos.



```
/*
 * InterfaceEnv.java 05/01/2004
 */

import java.rmi.*; // classe que possibilita o acesso remoto.

/* definição da assinatura da interface
 remota(IEEE1451_BaseTransducerBlock) */
public interface InterfaceEnv extends Remote {

    // definição dos métodos remotos
    public String EnviaMensagem(String mensagem) throws RemoteException;
}
```

Figura 4.10: Código fonte do programa *InterfaceEnv*.

As interfaces remotas não possuem o método *main()*, pois as interfaces não são executáveis. Porém, os métodos determinados nas interfaces são executáveis através do programa que implementa estas interfaces.

O programa *ServidorEnv* deve implementar a interface remota (*InterfaceEnv*) e receber um objeto remoto. O *ServidorEnv* cria um objeto denominado de *EnviaDado* para possibilitar que os clientes encontrem os seus serviços. Na Figura 4.11 apresenta-se o código fonte do programa responsável por implementar as funcionalidades de um servidor.


```

ServidorEnv - Bloco de notas
Arquivo  Editar  Pesquisar  Ajuda

/**
 * ServidorEnv.java 05/01/2004
 */
// Conjunto de classes requeridas para compilar o programa.
import java.rmi.*; // classe que possibilita o acesso remoto.
import java.rmi.server.*; // classe que implementa o servidor remoto
// Assinatura do Servidor
public class ServidorEnv extends UnicastRemoteObject
    implements InterfaceEnv {
    public ServidorEnv() throws RemoteException {
    }

    // Método recebe a mensagem de um objeto remoto.
    public String EnviaMensagem(String mensagem) {
        System.out.println("O cliente enviou a seguinte mensagem: " + "\n" + mensagem);
        return mensagem;
    }

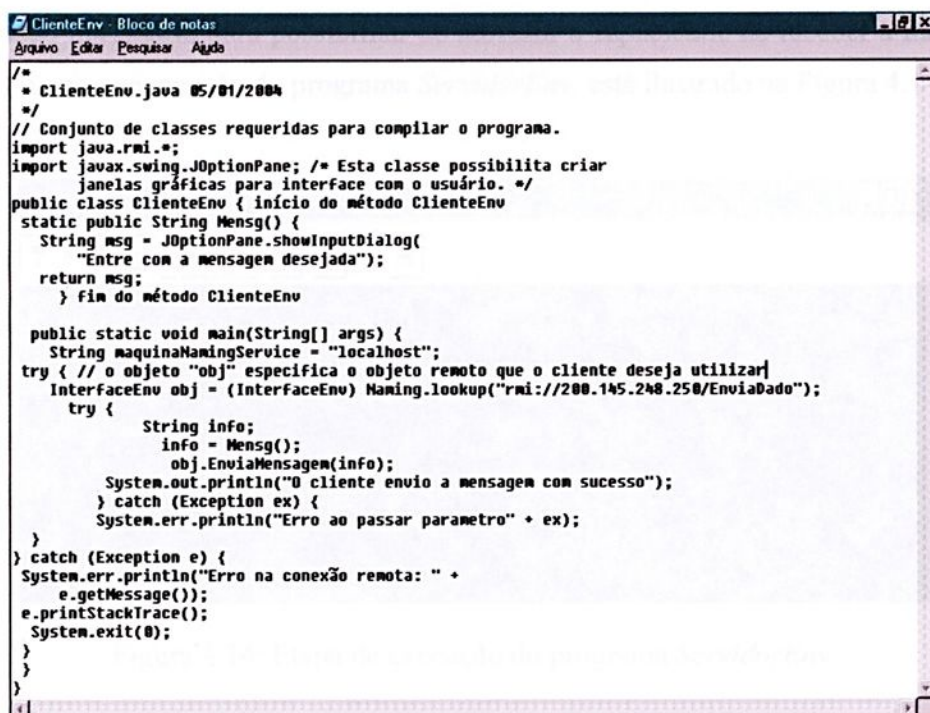
    public static void main(String[] args) {
        try {
            //assinatura do objeto remoto do servidor.
            String names = "//localhost/EnviaDado"; // objeto remoto EnviaDado
            InterfaceEnv eu = new ServidorEnv();
            // Registra o serviço
            Naming.rebind(names, eu); // vincula o objeto remoto eu ao servidor remoto.

            System.out.println("Servidor a espera da mensagem");
            System.out.println("Servidor pronto para receber mensagem...");
        } catch (Exception e) { // declaração do tipo de exceção para este trecho do programa.
            System.err.println("exception: " + e);
        }
    }
}

```

Figura 4.11: Código fonte do programa *ServidorEnv*.

O programa *ClienteEnv* desenvolve uma interface gráfica para o usuário inserir a mensagem que deseja enviar ao servidor, e sabe qual objeto remoto deseja utilizar. Na Figura 4.12 apresenta-se o código fonte do programa *ClienteEnv*.

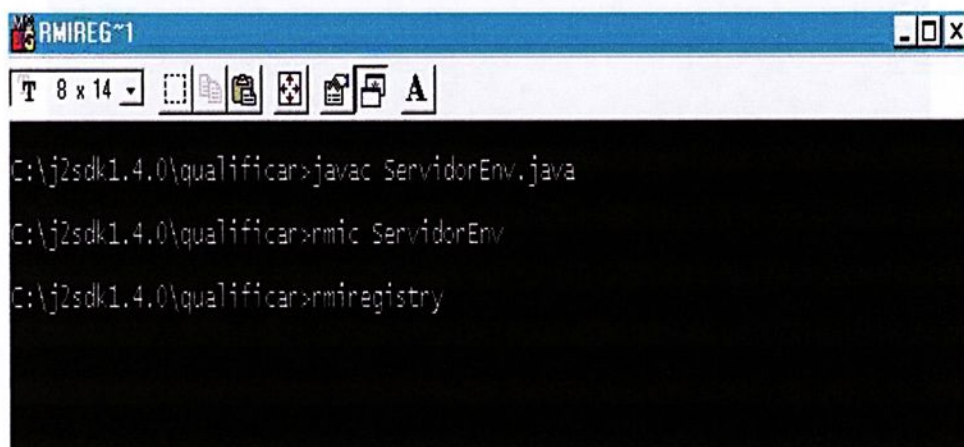


```
/*
 * ClienteEnv.java 05/01/2004
 */
// Conjunto de classes requeridas para compilar o programa.
import java.rmi.*;
import javax.swing.JOptionPane; /* Esta classe possibilita criar
janelas gráficas para interface com o usuário. */
public class ClienteEnv { início do método ClienteEnv
static public String Mensg() {
String msg = JOptionPane.showInputDialog(
"Entre com a mensagem desejada");
return msg;
} fim do método ClienteEnv

public static void main(String[] args) {
String maquinaNamingService = "localhost";
try { // o objeto "obj" especifica o objeto remoto que o cliente deseja utilizar
InterfaceEnv obj = (InterfaceEnv) Naming.lookup("rmi://200.145.248.250/EnviaDados");
try {
String info;
info = Mensg();
obj.EnviaMensagem(info);
System.out.println("O cliente envio a mensagem com sucesso");
} catch (Exception ex) {
System.err.println("Erro ao passar parametro" + ex);
}
} catch (Exception e) {
System.err.println("Erro na conexão remota: " +
e.getMessage());
e.printStackTrace();
System.exit(0);
}
}
}
```

Figura 4.12: Código fonte do programa *ClienteEnv*.

Na Figura 4.13 apresenta-se as etapas necessárias para compilação dos programas *ServidorEnv* e *ClienteEnv*, bem como, a etapa de registro do objeto remoto na rede de comunicação.



```
C:\j2sdk1.4.0\qualificar>javac ServidorEnv.java
C:\j2sdk1.4.0\qualificar>rmic ServidorEnv
C:\j2sdk1.4.0\qualificar>rmiregistry
```

Figura 4.13: Etapas de compilação e registro do *ServidorEnv*.

O comando necessário para possibilitar ao servidor a capacidade de receber a mensagem de um cliente, ou seja, a execução do programa *ServidorEnv*, está ilustrado na Figura 4.14.

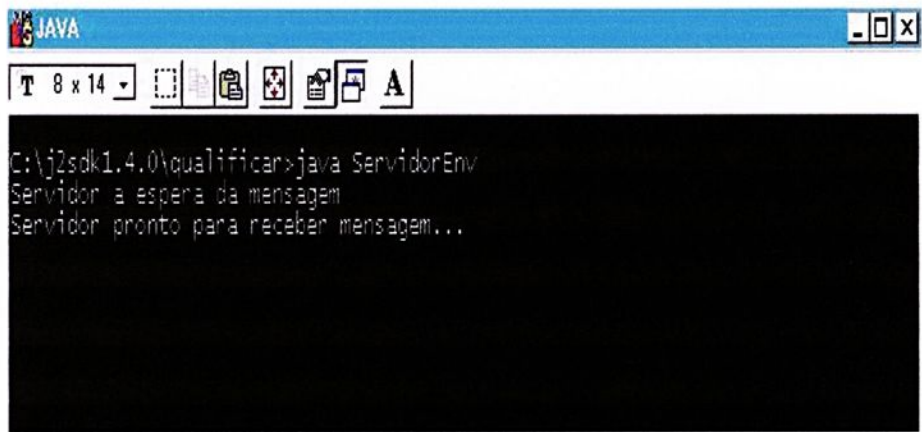


Figura 4.14: Etapa de execução do programa *ServidorEnv*.

Na Figura 4.15 mostra-se as etapas necessárias para compilar e executar o programa *ClienteEnv*. Ao executar o programa *ClienteEnv*, abre-se uma janela gráfica, permitindo-se ao cliente inserir uma mensagem. O usuário deve clicar no ícone [ok] para enviar a mensagem ao servidor.

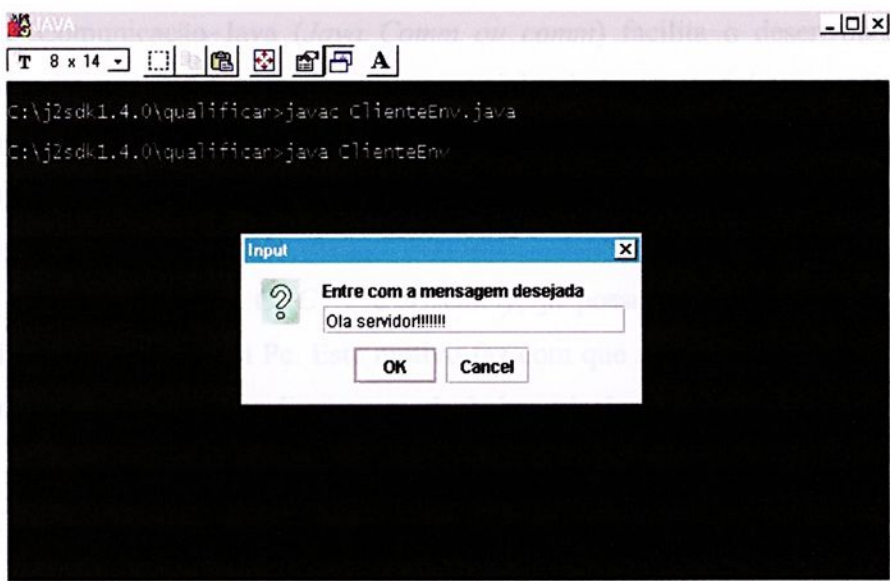


Figura 4.15: Etapas para compilar e executar o programa *ClienteEnv*.

Na Figura 4.16 mostra-se o servidor (*ServidorEnv*) recebendo a mensagem enviada pelo cliente (*ClienteEnv*).

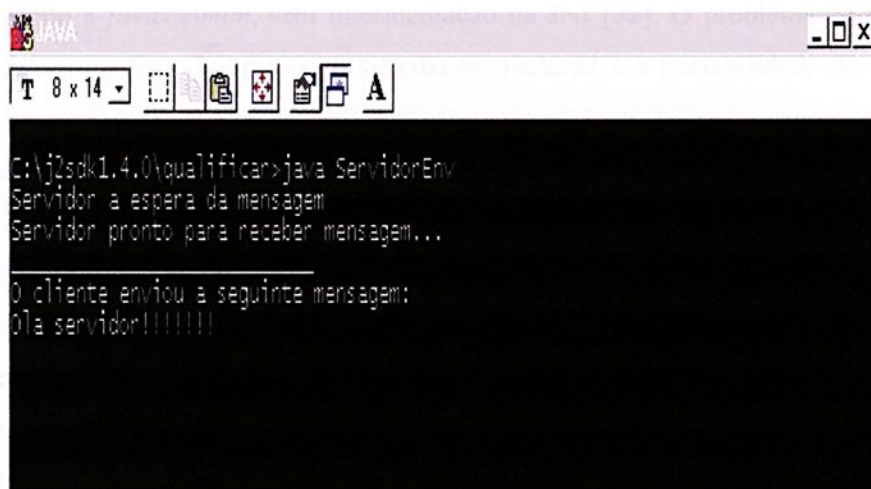


Figura 4.16: O servidor recebe a mensagem de um cliente.

4.8.8 API de Comunicação Java

A API de Comunicação Java (*Java Comm* ou *comm*) facilita o desenvolvimento de software para controlar dispositivos através da porta serial e da porta paralela de um Pc [31]. Ou seja, a *Java Comm* possibilita o emprego de Java em baixo nível de programação²⁷, sendo uma ferramenta interessante para desenvolver software aplicado na automação industrial e automação para o consumidor final.

Algumas linguagens, como *C*, *C++* e *Assembly*, já possuem diversas aplicações para controlar dispositivos através de um Pc. Este motivo faz com que alguns programadores, muitas vezes, criem um programa em uma linguagem de baixo nível como, por exemplo, *C* e, em seguida, através da JNI (*Java Native Interface*), passe para Java. Porém, apesar do Brasil ser considerado a maior comunidade de usuários de Java do mundo, há uma carência de

²⁷ Baixo nível de programação: são programas que implementam linguagens de máquina.



implementações nos fóruns de Java e na literatura, que executam o controle de atividades de dispositivo empregando a *javax.comm*, sem intermediação da JNI [32]. O problema de utilizar este recurso, é que o aplicativo fica preso ao sistema operacional e a portabilidade do Java é perdida.

A *Java Comm* é destinada para enviar/receber dados através da porta paralela e da porta serial de um Pc. Porém, uma aplicação usando Java como linguagem de baixo nível, é algo inexplorado atualmente, uma vez que a maioria dos programadores trabalha com Java em alto nível.

As vantagens de empregar Java como linguagem de baixo nível, está nos recursos que a linguagem oferece para aplicações remotas e distribuídas, além de possuir a independência com plataformas, o que constitui uma ferramenta extremamente útil nas áreas de instrumentação distribuída, construção de sistemas embarcados, implementação de laboratório à distância (uso de equipamentos via Internet), etc. Pensando no consumidor final, a *comm* possibilita, por exemplo, o controle integrado de eletrodomésticos através de um único Pc. E, como o uso de computador pessoal em conjunto com a Internet vêm se tornando cada vez mais popular, pode-se prever um aumento maciço na utilização da *Java Comm* nas residências e nas indústrias.

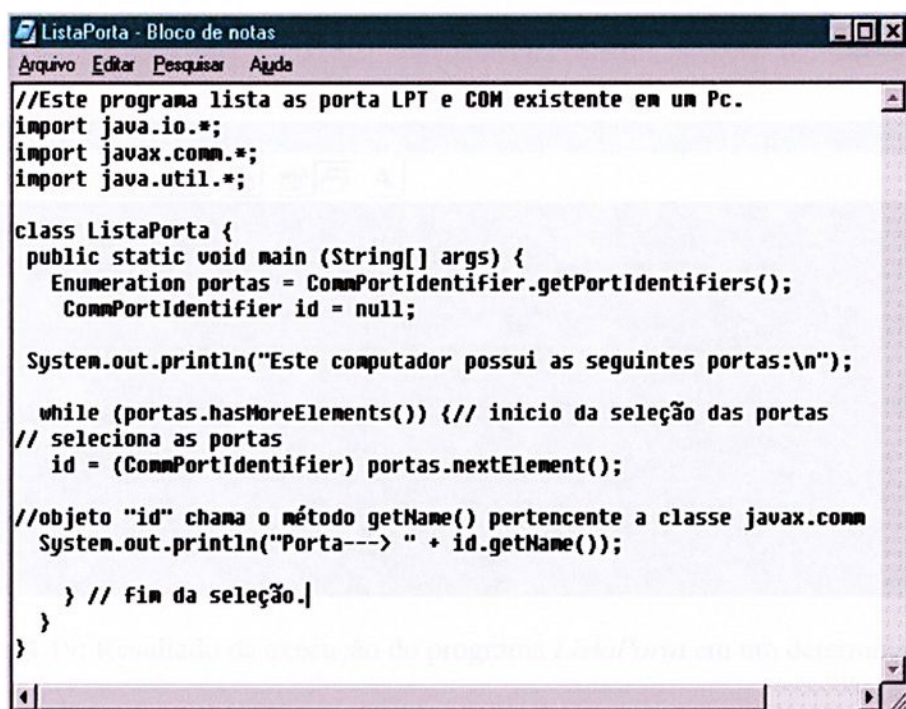
A *Java comm* não está inserido no *jdk* de nenhuma versão, portanto, para obtê-la é necessário realizar um download da API e instalá-la corretamente.

No site com endereço <http://java.sun.com/products/javacomm/>, encontra-se a API *Comm* a qual pode ser obtida gratuitamente. Dentro do pacote encontra-se uma pasta que descreve o funcionamento de toda a classe e interface pertencentes a *Comm*, exemplos de aplicação e um roteiro para instalação do aplicativo. No Apêndice C apresenta-se a descrição de algumas classes e interfaces pertencentes ao pacote *javacomm*. Na Figura 4.17 descreve-se o código fonte de um programa que utiliza o pacote *javacomm*. Este programa, denominado de *ListaPorta*, cria uma lista de portas do tipo *LPT*²⁸ e *COM*²⁹ existente em um microcomputador. O *ListaPorta* utiliza a classe *CommPortIdentifier()* para chamar o método *getPortIdentifier()* e identificar as portas existentes no Pc. Já o método *getName()* fornece o nome da porta identificada. A etapa *System.out.println("....." + id.getName());* envia para saída padrão (monitor) o nome da porta indentificada.

²⁸ LPT: refere-se às portas paralelas de um Pc.

²⁹ COM: refere-se às portas seriais de um Pc.





```
//Este programa lista as porta LPT e COM existente em um Pc.
import java.io.*;
import javax.comm.*;
import java.util.*;

class ListaPorta {
    public static void main (String[] args) {
        Enumeration portas = CommPortIdentifier.getPortIdentifiers();
        CommPortIdentifier id = null;

        System.out.println("Este computador possui as seguintes portas:\n");

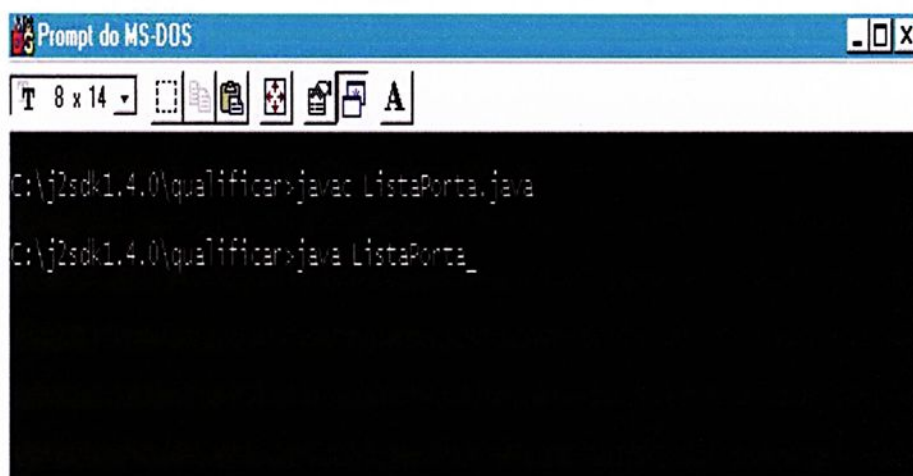
        while (portas.hasMoreElements()) { // inicio da seleção das portas
            // seleciona as portas
            id = (CommPortIdentifier) portas.nextElement();

            //objeto "id" chama o método getName() pertencente a classe javax.comm
            System.out.println("Porta---> " + id.getName());

            } // fim da seleção.
        }
    }
}
```

Figura 4.17: Código fonte de um programa que utiliza o pacote *javax.comm*.

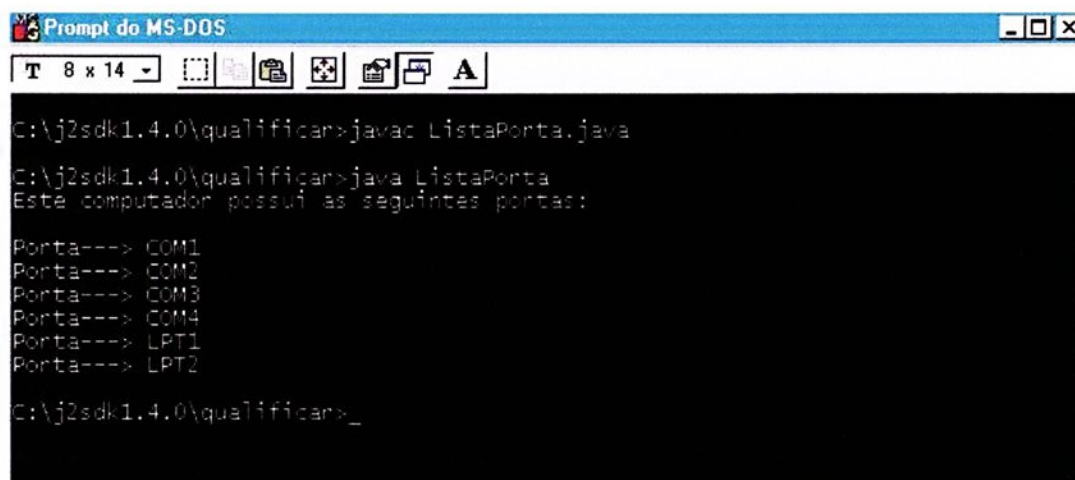
Na Figura 4.18 mostram-se as etapas necessárias para compilar e executar o programa *ListaPorta*.



```
C:\j2sdk1.4.0\qualifican>javac ListaPorta.java
C:\j2sdk1.4.0\qualifican>java ListaPorta_
```

Figura 4.18: Etapas para compilar e executar o programa *ListaPorta*.

Na Figura 4.19 ilustra-se a lista de portas existente em um Pc.



```
Prompt do MS-DOS

T 8 x 14

C:\j2sdk1.4.0\qualifican>javac ListaPorta.java
C:\j2sdk1.4.0\qualifican>java ListaPorta
Este computador possui as seguintes portas:

Porta---> COM1
Porta---> COM2
Porta---> COM3
Porta---> COM4
Porta---> LPT1
Porta---> LPT2

C:\j2sdk1.4.0\qualifican>_
```

Figura 4.19: Resultado da execução do programa *ListaPorta* em um determinado Pc.

CAPÍTULO - 5

5. Padrão IEEE1451

5.1 Introdução

Os transdutores, designação usada aqui para se referir a sensores e/ou atuadores, possuem uma grande variedade de aplicações, como: em linha de produção, na indústria aeroespacial, em sistemas de segurança, em casas e edifícios inteligentes, em engenharia biomédica, etc. Seguindo a tendência da tecnologia, os fabricantes de transdutores procuram maneiras de construir transdutores inteligentes, com capacidade de trabalhar em rede e com o menor custo possível. Muitas redes de controle de transdutores que estão disponíveis atualmente são proprietárias. As redes proprietárias possuem baixa flexibilidade e restringem o cliente aos serviços de apenas um determinado fabricante, causando o encarecimento desse serviço. A tarefa de conectar os transdutores inteligentes a uma rede de controle que suporta a grande variedade de protocolos, ou seja, redes abertas, exigem esforços significativos, pois aumenta a complexidade do sistema de automação. Entretanto, a indústria torna-se mais eficiente e com grande possibilidade de reduzir os custos operacionais. Portanto, um ponto próximo do ideal para transdutores inteligentes trabalharem em rede, seria um sistema totalmente aberto e padronizado. Com esse objetivo surge o padrão proposto pelo comitê do IEEE 1451 [33].

5.2 Objetivo do padrão IEEE 1451

O objetivo do IEEE 1451 é desenvolver uma interface padronizada para conectar transdutores em qualquer tipo de rede de comunicação. Assim, a interface padronizada deve



utilizar as tecnologias de redes de controles existentes e suportar os diferentes tipos de sensores e atuadores, tornando estes dispositivos inteligentes [34].

5.3 História do padrão IEEE1451

Em setembro 1993, o NIST e os coordenadores da Sociedade de Tecnologia de Sensores e Instrumentação e Medidas, comitê técnico do (IEEE), patrocinaram uma reunião para discutir a possibilidade de criar uma interface padrão para comunicação entre sensores inteligentes conectados em rede. Desta reunião, originaram-se quatro grupos de pesquisa distintos: IEEE1451.1, IEEE1451.2, IEEE1451.3 e IEEE1451.4. Cada grupo ficou responsável por desenvolver uma parte do padrão [35]. Na Figura 5.1 apresenta-se a configuração de um transdutor inteligente em conformidade com o padrão IEEE1451, proposto em 1993. O padrão IEEE1451.1 é a interface comum para os demais padrões [36].

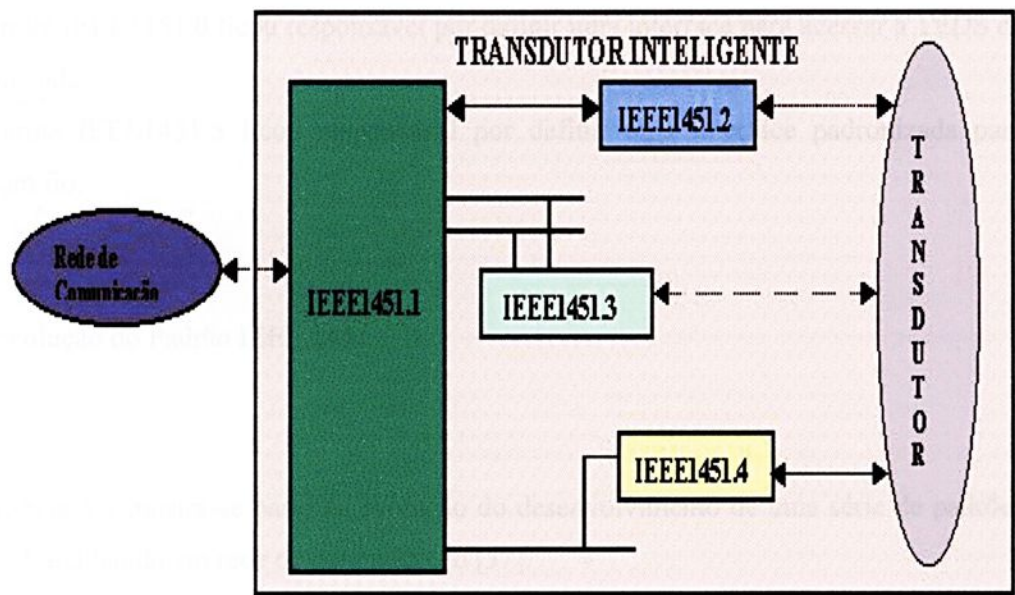


Figura 5.1: Maneira como a família de padrões IEEE 1451 estão relacionados.

Por volta de 2002, surgiram os comitês IEEE1451.0 e IEEE1451.5.

O grupo de pesquisa P1451.1 ficou responsável por definir um processador que desempenhasse as funções de um nó de rede³⁰. O processador, denominado de processador de aplicação com capacidade de operar em rede de comunicação (*Network Capable Application Processor* - NCAP), devia possuir interoperabilidade para proporcionar inteligência ao transdutor

O grupo de pesquisa P1451.2 ficou responsável por definir um módulo de interface para transdutor inteligente (STIM), um bloco de dados eletrônicos do transdutor (*Transducer Electronic Data Sheet* - TEDS) e uma interface independente do transdutor (*Transducer Independent Interface* - TII). Este grupo propôs que o STIM devesse possuir uma capacidade *plug and play* (conecte e opere) com a TII.

O grupo de pesquisa P1451.3 ficou responsável por produzir uma interface de comunicação digital para sistemas distribuídos, o Módulo de Interface para Transdutores conectados em Barramento (*Transducer Bus Interface Modules* - TBIM).

O grupo de pesquisa P1451.4 visou definir um protocolo de comunicação denominado de modo misto, isto é, para transmitir sinais analógicos e sinais digitais através do mesmo meio físico denominado de *Mixed-Mode Transducer*.

O comitê IEEE1451.0 ficou responsável por definir uma interface para acessar a TEDS de forma padronizada.

O comitê IEEE1451.5 ficou responsável por definir uma interface padronizada para aplicações sem fio.

5.4 Evolução do Padrão IEEE 1451

Na Tabela 5.1 mostra-se parte da evolução do desenvolvimento de uma série de padrões para sensores trabalhando em rede de comunicação [37].

³⁰ Um nó de rede é composto de hardware e software para implementar a comunicação em rede.



Tabela 5.1: Evolução do padrão IEEE1451.

Setembro de 1993	Surge a discussão para desenvolver uma interface para sensores inteligentes (Filadélfia, EUA).
31 março a 1 abril 1994	Primeira série de seminários sobre padronização de uma interface para sensores inteligentes, promovidas pelo IEEE/NIST (Gaithersburg, EUA).
9 setembro 1994	Segunda série de seminários, com a demonstração do conceito da TEDS e a formulação de dois grupos de pesquisa: 1415.1 e 1451.2 (Cleveland, EUA).
18 – 19 maio 1995	Na terceira série de seminários, apresenta-se a idéia de um STIM, que incluísse as características dos sensores em formato eletrônico, o TEDS. Foi mostrada a comunicação com um NCAP através de uma interface comum e foi apresentado o modo <i>plug-and-play</i> de vários sensores conectados em diferentes tipos de redes (Boston, EUA).
14 setembro 1995	A quarta série de seminários, “Sensor Expo”, foi realizada em (Chicago, EUA).
15 a 17 Novembro 1995	Na quinta série de seminários apresentou-se o aspecto técnico do padrão proposto. Realizou-se a demonstração de uma execução do modelo de referência, de uma aplicação dos transdutores inteligente, utilizando uma proposta de interface padronizada.
21 junho 1996	Primeira série de seminários sobre o padrão P1451.3 em (Gaithersburg, EUA).
30 Agosto 1996	Votação de um esboço do padrão 1451.2 no IEEE.
Setembro 1996	Foi formado o comitê para construir um esboço dos padrões IEEE1451.3 e IEEE 1451.4.
22 Outubro 1996	Exposição dos esboços das especificações para o 1451.1, 1451.2, 1451.3 e 1451.4 na (Filadélfia, EUA).
28 Fevereiro 1997	Votação para aprovação do padrão P1451.1, no IEEE.
26 Setembro 1997	O IEEE define o padrão IEEE P1451.2.
28 Setembro 1999	O IEEE define o padrão IEEE P1451.1.
1999/2000	Aparecem as primeiras placas comerciais para suportar um NCAP.
Outubro/2001	Workshop dos 100 anos do NIST. Apresentação de trabalhos relacionados com o padrão IEEE1451 e proposta de modificações na implementação das TEDS.
2002/2003	Apresentam-se as primeiras idéias sobre IEEE P1451.0 (unificação das TEDS) e IEEE P1451.5 (padrão IEEE1451 para rede sem fio).
2004	Aprovado o padrão IEEE1451.4

Os padrões 1451.2 e 1451.1 já estão bem definidos, enquanto os padrões 1451.0, 1451.3, 1451.4 e 1451.5 estão ainda em fase de desenvolvimento até a presente data. Em 2004 foi aprovada a primeira versão do padrão IEEE1451.4. Por este motivo, o texto apresenta uma análise somente dos padrões (IEEE1451.2 e IEEE1451.1) já concluídos.



5.5 O padrão IEEE1451.2

O padrão IEEE 1451.2 determina as funções do STIM e da TII.

5.5.1 STIM - Módulo Interface para Transdutores Inteligentes

O STIM pode ser composto por 1 a 255 sensores ou atuadores, ou mesmo a combinação desses. O STIM consiste de circuitos de condicionamento de sinais, um conversor A/D³¹ e/ou conversor D/A³², um microcontrolador, um bloco TEDS e um circuito para implementar a lógica de endereçamento. Esta lógica de endereçamento facilita a comunicação entre o STIM e o NCAP e pode ser realizada pelos microprocessadores [38]. Quando um STIM contém mais de um transdutor, pode ser denominado de STIM multicanal. Cada transdutor recebe um endereço de 1 a 255 e o controle de endereçamento é realizado pelo NCAP.

Quando o STIM é acionado, a informação dentro do TEDS está disponível ao NCAP para uso local e/ou para distribuição destas informações na rede. Portanto, o STIM é controlado pelo NCAP.

a) Microcontrolador

O microcontrolador tem a função, no STIM de gerenciar as tarefas solicitadas pelo NCAP. O microcontrolador pode dividir as tarefas em módulos, e assim, cada módulo terá uma função específica, ou seja, um módulo será responsável pelo tratamento dos sinais recebidos através do NCAP, outro módulo será responsável pelo tratamento de dados enviados ao NCAP, outro realizará a leitura de dados no TEDS, etc. Dessa forma pode facilitar ao projetista, o

³¹ A/D: Analógico/Digital, conversão de um sinal analógico para digital.

³² D/A: Digita/Analógico, conversão de um sinal digital para analógico.



desenvolvimento do protocolo de comunicação do STIM. O padrão 1451.2 define quais as tarefas que deverão ser executadas pelo microcontrolador.

b) TEDS

O TEDS foi a primeira inovação tecnológica inserida pelo P1451.2. O TEDS é responsável pelo armazenamento das características gerais dos transdutores como:

- a) dados dos fabricantes do transdutor;
- b) endereço do transdutor;
- c) faixa de atuação;
- d) informação para calibração, etc.

Essas características e informações são guardadas em forma eletrônica em um dispositivo de memória não volátil como, por exemplo, um chip de memória *EPROM*³³. Assim, quando o cliente deseja analisar o estado de um transdutor, ou mesmo, realizar um ajuste em um dispositivo, estas operações serão realizadas através do protocolo de comunicação, passando pelo TEDS, via microcontrolador que, por sua vez, passa pela TII, pelo NCAP e, finalmente, chega até o cliente. O TEDS pode ser dividido nos seguintes campos de dados: Meta-TEDS, canal-TEDS, calibração-TEDS, aplicação-TEDS e extensão-TEDS.

b1) Meta-TEDS

O Meta-TEDS é o campo de dados comum a todos os transdutores conectados ao STIM. Contém uma descrição total da estrutura dos transdutores.

³³ EPROM: Erasable Programmable Ready-Only Memory – memória de leitura apagável e programável.



b2) Canal-TEDS

O Canal-TEDS contém a informação sobre a localização dos transdutores (1 a 255), ou seja, mostra o endereço dos dispositivos.

b3) Calibração-TEDS

A Calibração-TEDS contém informações sobre a última calibração, os parâmetros da calibração e o intervalo da calibração de um transdutor, além de fornecer também as constantes e os métodos necessários para converter os dados crus em unidades de engenharia. Este serviço é opcional. O padrão não especifica como ou qual software deve ser utilizado para realizar a calibração. Porém, o padrão IEEE1451.2 define que a calibração deve ser realizada por meio de uma metodologia de domínio aberto.

b4) Aplicação-TEDS

A Aplicação-TEDS contém informação que é específica da aplicação, variando de acordo com o uso do transdutor.

b5) Extensão-TEDS

A extensão-TEDS é reservada para implementações futuras e expansões industriais, em conformidade com padrão IEEE1451.2.

5.5.2 TII – Interface Independente do Transdutor



A TII é a interface responsável pela comunicação física entre o NCAP e o STIM. Esta comunicação consiste de 10 pinos bem definidos. O conjunto de pinos faz com que seja possível estabelecer o protocolo de comunicação entre o STIM e o NCAP. Na Tabela 5.2 apresenta-se a direção com que é realizada a troca de informações e a padronização da interface digital TII.

Tabela 5.2: Pinagem recomendada para a TII.

Pinagem	Sinal	Cor do Fio	Para NCAP	para STIM
1	DCLK	Marrom	Out	In
2	DIN	Vermelho	Out	In
3	DOUT	Laranja	In	Out
4	NACK	Amarelo	In	Out
5	TERRA	Verde	Power	Power
6	NIOE	Azul	Out	In
7	NINT	Violeta	In	Out
8	NTRIG	Cinza	Out	In
9	ALIMENTAÇÃO	Branco	Power	Power
10	NSDET	Preto	In	Out

Na Tabela 5.3 apresenta-se, de forma resumida, as funções de cada pino, como propostas pelo comitê P1451.2 para a TII.



Tabela 5.3: Descrição da pinagem da TII.

Pino	Nível Lógico	Controle	Função
DIN	Positivo	NCAP	Transportar endereços e dados do NCAP para o STIM
DOUT	Positivo	STIM	Transportar dados do STIM para o NCAP
DCLK	Borda de subida	NCAP	A cada pulso é recebido/armazenado um bit de dado
NIOE	Ativo baixo	NCAP	Sinalização de que o transporte é ativo e delimita o dado
NTRIG	Borda de descida	NCAP	Desempenha funções de disparo
NACK	Borda de descida	STIM	Reconhecimento de disparo e de transporte de dados
NINT	Borda de descida	STIM	Usado pelo STIM para requisitar serviço do NCAP
NSDET	Ativo baixo	STIM	Usado pelo NCAP para detectar a presença do STIM
POWER	N/A	NCAP	Alimentação 5V
COMUM	N/A	NCAP	Comum ou terra

5.6 O padrão IEEE 1451.1 - NCAP

O NCAP é responsável pela troca de informação ocorrida entre o transdutor e o ambiente externo, que pode ser um outro transdutor ou uma rede na qual está conectado. Para desempenhar estas atividades, o NCAP deve possuir capacidade de controlar o STIM e comunicar-se com uma rede de controle como, por exemplo, a Ethernet [39]. O NCAP é basicamente um nó de rede, composto de partes lógicas e partes físicas.

Na primeira apresentação de uma arquitetura para o NCAP, o comitê IEEE1451.1 definiu os seguintes componentes:

- a) *Network Hardware* (drive de rede): conexão física da rede com o NCAP;
- b) *Network Protocol* (protocolo de rede): software responsável por gerenciar o tipo de rede que o NCAP está trabalhando;



- c) *Network Protocol Logical Interface Specification* (Interface de especificação lógica de um protocolo de rede): software responsável por realizar a lógica de interfaceamento entre o protocolo da rede e o Software de Aplicação;
- d) *Application Software* (Software de Aplicação): software responsável por gerenciar a comunicação entre o protocolo de rede e o protocolo do transdutor;
- e) *Transducer Logical Interface Specification* (Interface de especificação lógica de um transdutor): software responsável por desempenhar o protocolo de comunicação de um transdutor;
- f) *Transducer Software* (Software para Transdutor): software responsável por implementar o controle das atividades de um transdutor;
- g) *I/O Port Hardware* (Porto³⁴ Físico de Entrada/Saída): responsável por permitir a comunicação entre o NCAP e o transdutor;
- h) *Transducer Hardware Interface Specification* (Interface específica para o Transdutor): software responsável por gerenciar as atividades de um transdutor.

Na Figura 5.2 mostra-se a arquitetura originalmente proposta por um NCAP.

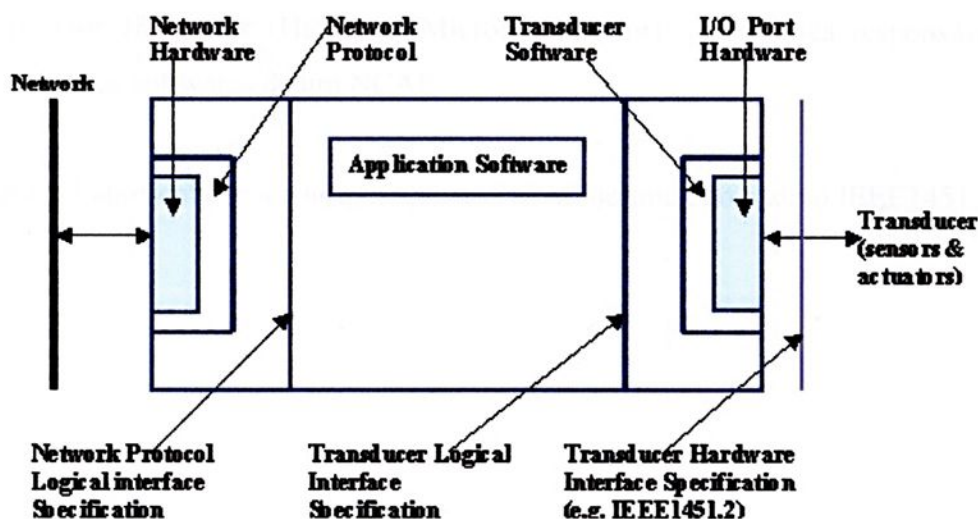


Figura 5.1: Primeira configuração de um NCAP

³⁴ Porto: neste caso, é o módulo ou bloco.

Com objetivo de fornecer maiores informações sobre as funções de um NCAP, o comitê IEEE1451.1 decidiu apresentar uma configuração mais detalhada, principalmente, na especificação do software de aplicação. Os componentes que completaram a configuração final do NCAP foram:

- a) *Portos, Network Abstraction Logical Interface Specification* (Módulo, Especificação de uma Interface Lógica para Abstrair o tipo de Rede de Comunicação): módulo que deve possibilitar que o NCAP interprete os vários tipos de rede de controle;
- b) *Server Object Dispatch* (Expedição de Objeto para o Usuário): bloco destinado ao objeto que interpreta os tipos de rede de controle para o usuário;
- c) *Logical Interface to NCAP support* (Sustentação de uma Interface Lógica para o NCAP): Interface responsável por permitir o funcionamento sincronizado dos diferentes blocos lógicos de um NCAP;
- d) *NCAP Block* (Bloco NCAP): este bloco lógico deve controlar o funcionamento de todos componentes softwares pertencente ao NCAP;
- e) *Operating System Firmware*³⁵ (Sistema Operacional): bloco lógico responsável por permitir o funcionamento dos blocos software de um NCAP;
- f) *Microprocessor Hardware* (Hardware Microprocessador): parte física responsável por acoplar os blocos softwares de um NCAP.

Na Figura 5.3 apresenta-se a configuração do NCAP definida no padrão IEEE1451.1.

³⁵ Firmware: é a fase de inicialização da máquina, também chamado de boot. Firmware é a fase compreendida entre ligar o computador até carregar o sistema operacional.



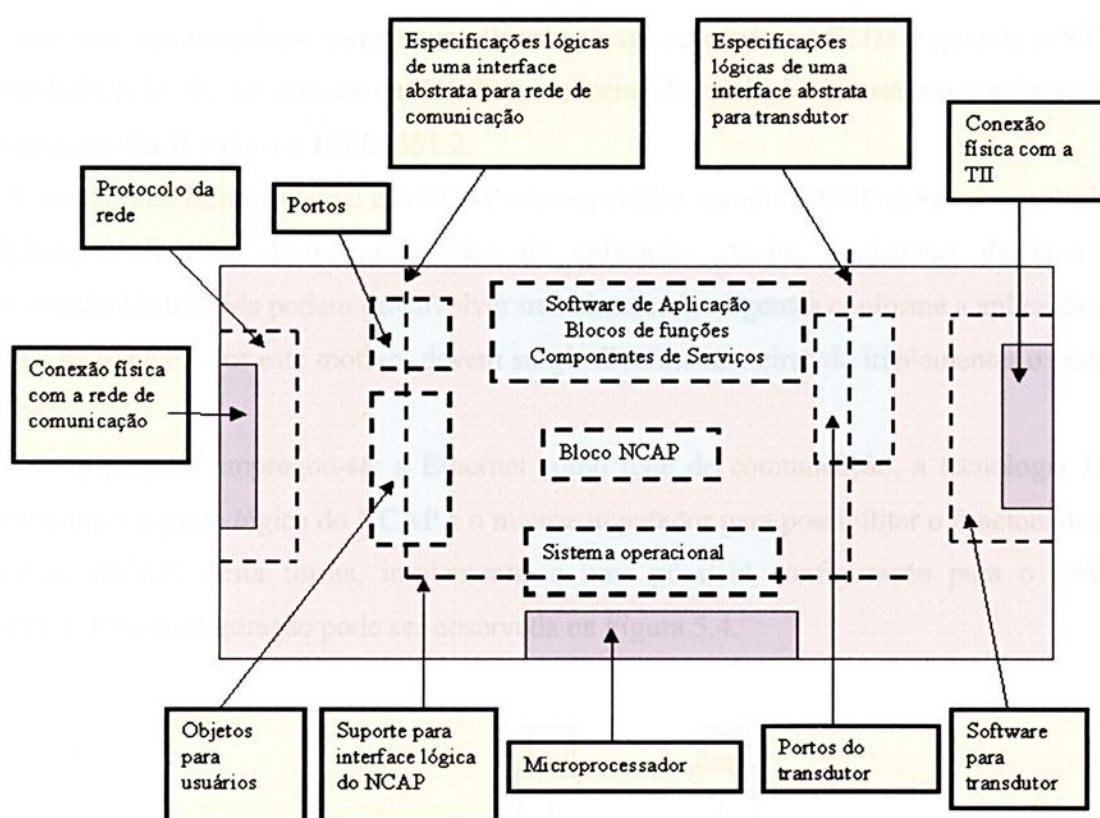


Figura 5.3: Configuração de um NCAP, em conformidade com o padrão IEEE1451.1.

No aspecto funcional, o modelo de informação define três características de um NCAP:

- Interoperabilidade:** A interoperabilidade é a característica pelo qual a interface NCAP pode ser conectada; em qualquer tipo de rede de controle. A interoperabilidade deve facilitar a conexão de transdutores nos diversos ambientes industriais;
- Modelagem Orientado a Objeto:** O modelo de informação é baseado em blocos que têm uma função ou várias funções. Os módulos de software devem possuir alto grau de abstração da aplicação. Desta forma as linguagens orientadas a objetos são ferramentas indispensáveis;
- Driver da Interface para Transdutor Inteligente:** Este hardware tem a função de possibilitar ao NCAP o controle do módulo para interface de transdutor inteligente (STIM).

Portanto, um transdutor torna-se inteligente ao ser acoplado ao STIM e quando o STIM for controlado pelo NCAP através da TII. Este conceito de inteligência está em conformidade com os padrões IEEE 1451.1 e IEEE1451.2.

O modelo de informação de um NCAP não especifica como o NCAP deve ser construído, mas define as funções deste processador de aplicação. Assim, projetistas da área de Instrumentação Distribuída podem desenvolver transdutores inteligentes conforme a aplicação e a tecnologia disponível. Por este motivo, devem surgir diversas maneiras de implementar o padrão IEEE1451.

Neste projeto, empregou-se: a Ethernet como rede de comunicação, a tecnologia Java para desenvolver a parte lógica do NCAP e o microcomputador para possibilitar o funcionamento do software NCAP, desta forma, implementa-se uma possível configuração para o padrão IEEE1451.1. Esta configuração pode ser observada na Figura 5.4.

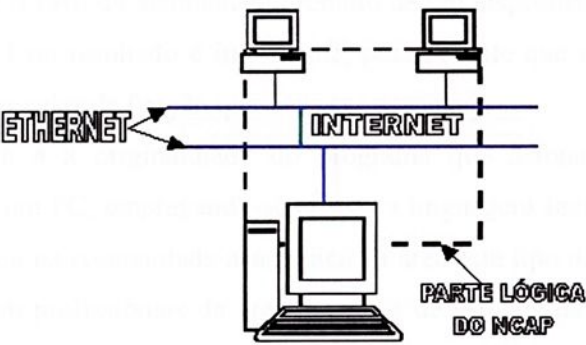


Figura 5.4: Configuração do projeto.

6. Resultados

6.1 Implementações

Atualmente, a implementação do módulo software do NCAP (software NCAP), conforme proposto neste trabalho, encontra-se em fase de testes. Alguns destes testes já mostraram resultados satisfatórios, como é o caso do acionamento remoto de um dispositivo de acordo com a determinação de um cliente. Este resultado é importante, pois permite que o Gerenciador de Protocolo receba de um cliente o valor da função que deve ser executada.

Outro aspecto relevante é a originalidade do programa que aciona um dispositivo conectado na porta paralela de um PC, empregando-se apenas a linguagem Java, uma vez que o grupo de pesquisa não encontrou na comunidade acadêmica da área este tipo de proposta. Além disso, ao entrar em contato com profissionais da área e grupos de estudos da linguagem Java, observou-se à carência de programas que utilizassem Java em baixo nível de programação. O software do NCAP, desenvolvido neste trabalho, oferece a possibilidade de um cliente enviar, via internet, o valor de uma função ao Gerenciador de Protocolo, o qual é composto de três programas:

- a) *IEEE1451_BaseTransducerBlock*: exerce as funções de uma interface na arquitetura RMI, definindo os métodos *IEEE1451NCAP()* e *recebeIP()*;
- b) *IEEE1451_FunctionBlock*: fornece o IP do cliente ao servidor, possui um menu com os endereços de funções e produz uma interface gráfica para que o cliente insira a função desejada e envie ao servidor;
- c) *IEEE1451_NCAPBlock*: responsável pela implementação da interface remota, possibilita a comunicação com o dispositivo conectado na porta paralela e deve detectar qual cliente está utilizando o seu serviço.



A estrutura funcional do software baseia-se na hierarquia de classe, definida no modelo de informação IEEE1451.1, como pode ser observado no Apêndice D.

A parte física do NCAP é composta de um PC e um Gerenciador de Protocolo. O PC deve possuir uma configuração suficiente para implementar a JVM e um editor de texto. No caso deste projeto, o PC possui um *clock* de 700MHz, com HD de 30Gb, e dotado do sistema operacional Windows Me. A porta paralela do microcomputador tem a função de acessar o Gerenciador de Protocolo. Conhecida como LPT ou porta Centronics, a porta paralela trabalha com os dados na forma de byte, e necessita apenas de um cabo para realizar a comunicação entre o computador e o dispositivo desejado [40]. A LPT possui os seguintes endereços de I/O³⁶ :

- a) 378h³⁷ registrador de dados de 8 bits;
- b) 379h registrador de estado de 5 bits;
- c) 37Ah registrador de controle de 4 bits.

O pacote Java COMM possui variáveis que implementam o endereçamento da porta desejada. Desta forma, quando o programador deseja utilizar a porta LPT, não necessita especificar o endereço da porta, como ocorre na linguagem de programação C, porém, deve informar o driver desejado. A porta paralela utilizada no projeto possui conector do tipo DB de 25 pinos. Na Figura 6.1 apresenta-se o conector DB-25 com seus respectivos registradores.

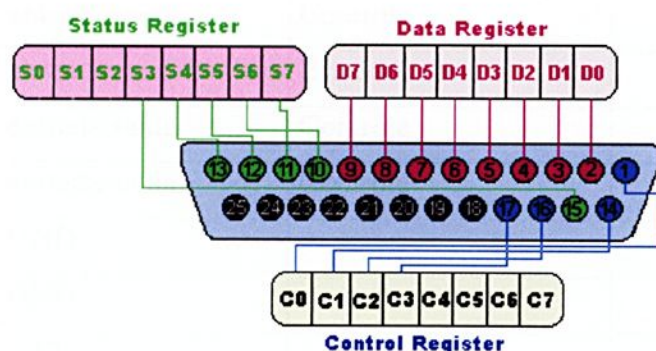


Figura 6.1: Conector DB-25. Fonte: <http://www.mrshp.hpg.ig.com.br/rob/paralela.htm>

³⁶ I/O: sigla de Input/Output, ou seja, Entrada/Saída.

³⁷ 378h: corresponde a um valor em hexadecimal.

A configuração da pinagem do conector DB-25 está descrita na Tabela 6.1.

Tabela 6.1: Pinagem do conector DB-25.

Pino	Dado Digital	Direção	Registrador associado	Ativo em nível lógico
1		entrada/saída	Controle	0
2	1	Saída	Dado	Lógica positiva
3	2	Saída	Dado	Lógica positiva
4	4	Saída	Dado	Lógica positiva
5	8	Saída	Dado	Lógica positiva
6	16	Saída	Dado	Lógica positiva
7	32	Saída	Dado	Lógica positiva
8	64	Saída	Dado	Lógica positiva
9	128	Saída	Dado	Lógica positiva
10	64	Entrada	Estado	0
11	128	Entrada	Estado	1
12	32	Entrada	Estado	1
13	16	Entrada	Estado	1
14	-	entrada/saída	Controle	0
15	8	Entrada	Estado	0
16	-	entrada/saída	Controle	0
17	-	entrada/saída	Controle	0
18	-	GND	-	-
19	-	GND	-	-
20	-	GND	-	-
21	-	GND	-	-
22	-	GND	-	-
23	-	GND	-	-
24	-	GND	-	-
25	-	GND	-	-



Para realizar a comunicação, a porta paralela atua em diferentes modos de transmissão como, por exemplo:

- a) *Enhanced Parallel Port* (EPP): suporta transmissão bidirecional, com capacidade para transmitir até 2 MB/s e utiliza 8 bits por vez para transmitir dados;
- b) *Enhanced Capabilities Port* (ECP): tem as mesmas características que a EPP, porém, utiliza DMA (acesso direto à memória);
- c) *Standard Parallel Port* (SPP): suporta transmissão unidirecional, pode transmitir dados até 150 KB/s e utiliza 4 bits por vez para transmitir dados.

O modo de transmissão da porta paralela é acionado através do Sistema Básico de Entrada e Saída (*Basic Input Output System* - BIOS) de cada PC. A saída da porta paralela apresenta, normalmente, nível lógico *TTL*³⁸ (para valores inferiores a 0,5 V nível lógico 0 e para valores de 3,25 V, nível lógico 1).

O Gerenciador de Protocolo, que está em fase de teste, foi desenvolvido em VHDL no ambiente MAX + Plus[®]II da Altera e está sintetizado em dispositivo FPGA. A função principal do Gerenciador de Protocolo é obter o comando de disparo via software e iniciar o processo que estabelece o protocolo IEEE 1451.2, controlando as atividades do STIM. Assim, o Gerenciador de Protocolo foi desenvolvido com o intuito de construir um dispositivo simples que possa ser empregado em múltiplas aplicações e possibilite o modo de operação *plug and play* do STIM. O STIM também foi desenvolvido em VHDL e também está sintetizado em FPGA. O software NCAP deve controlar todo o processo de troca de informação ocorrida entre os clientes e o STIM. O software proporciona aos clientes conectados em uma rede Ethernet, capacidade de enviarem via Internet, endereços de funções ao servidor. O servidor, por sua vez, pode estar localizado em uma outra rede Ethernet e deve acionar o Gerenciador de Protocolo, através da porta paralela do PC, conforme a função solicitada pelos clientes. Quando o Gerenciador de Protocolo recebe as instruções, executa uma lógica de endereçamento e em seguida envia através da TII, um comando para o STIM. Assim, o STIM aciona as atividades do transdutor conforme o

³⁸ TTL: *Transistor-Transistor Logic*



endereço de função solicitado pelos clientes. Na Figura 6.2 apresenta-se a configuração do projeto no qual o software NCAP deve gerenciar o controle dos processos.

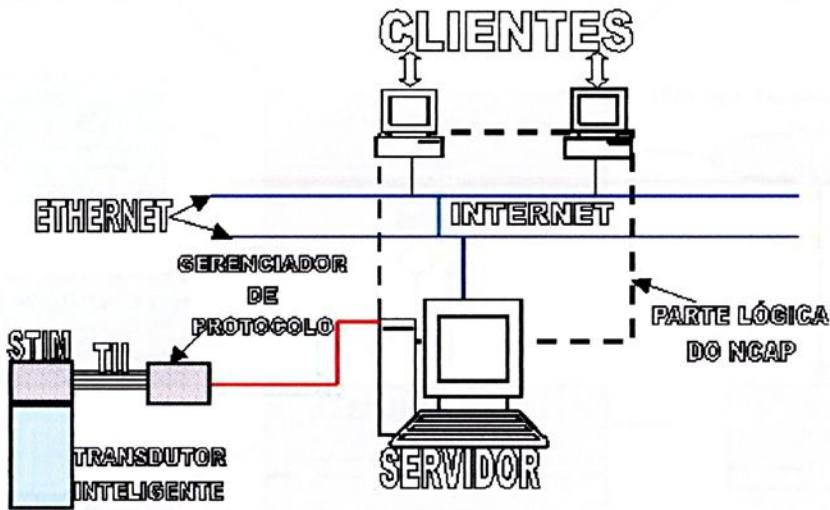


Figura 6.2: Configuração do projeto controlado pelo software NCAP.

6.2 Software NCAP

Nesta seção apresenta-se a descrição do código fonte do software NCAP. Na Figura 6.3, mostra-se através de recursos da Linguagem de Modelagem Unificada (do inglês *Unified Modeling Language*, abreviada como *UML*), o diagrama de classe para desenvolver o software proposto. Neste diagrama observa-se o relacionamento entre as classes e interfaces. A classe *IEEE1451_NCABlock* utiliza alguns métodos e variáveis das classes *Enumeration*, *ParallelPort*, *CommPortIdentifier* e *OutputStream*, herda o comportamento da classe *UnicastRemoteObject*, recebe uma coleção de parâmetros da classe *IEEE1451_FunctionBlock* e implementa a interface *IEEE1451_BaseTransducerBlock*. A interface *IEEE1451_BaseTransducerBlock* herda o comportamento da classe *Remote*.



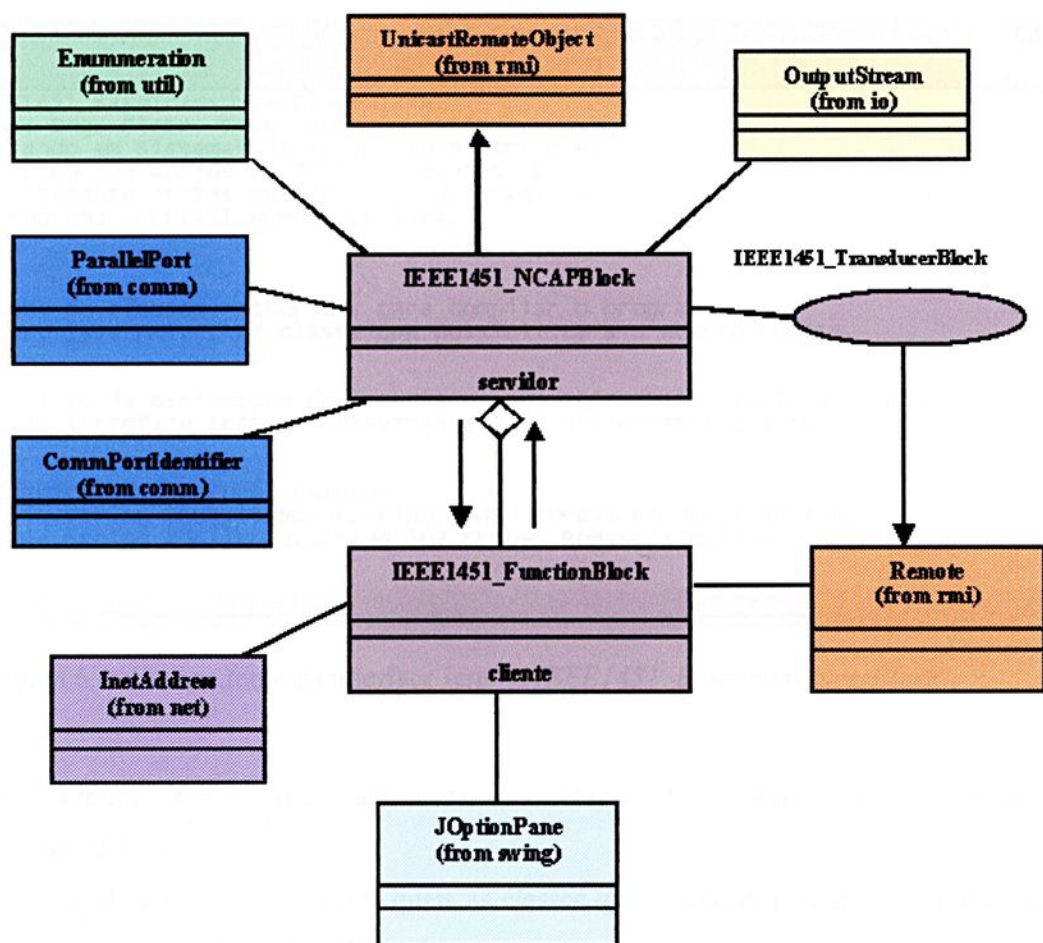
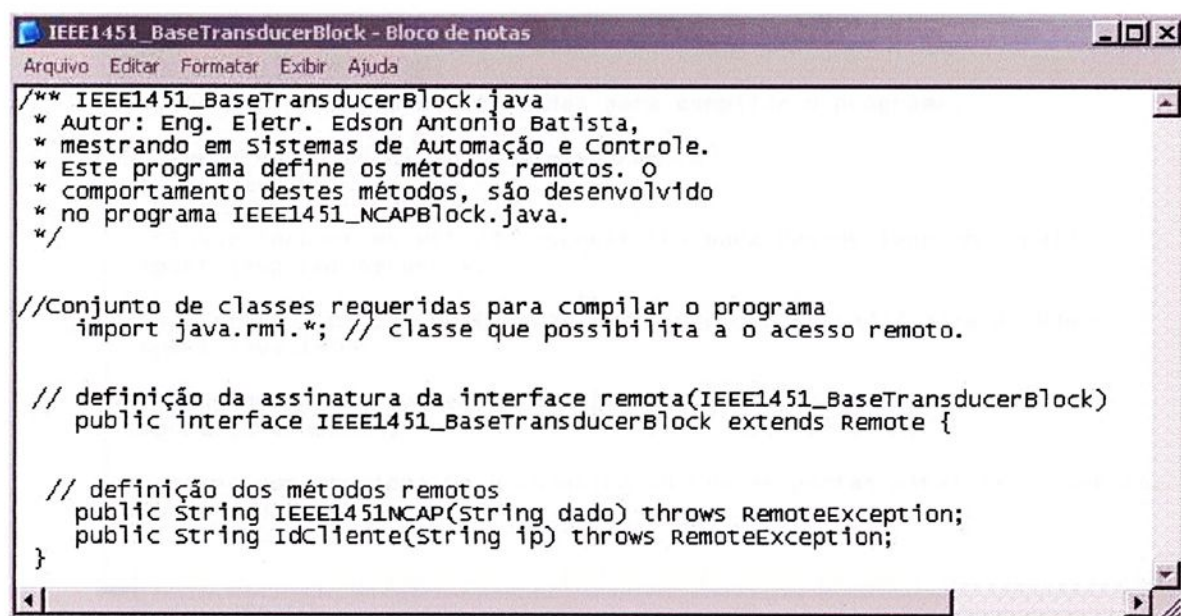


Figura 6.3: Diagrama de classe para desenvolver o software NCAP.

Pode-se observar, na Figura 6.4, a descrição de parte do código fonte do programa *IEEE1451_BaseTransducerBlock*. Neste programa, definem-se os métodos remotos, *IEEE1451NCAP()* e *IdCliente()*, ambos devem retornar um *String*, e desenvolvem suas funções no programa *IEEE1451_NCAPBlock*.



```
IEEE1451_BaseTransducerBlock.java
* Autor: Eng. Eletr. Edson Antonio Batista,
* mestrando em Sistemas de Automação e Controle.
* Este programa define os métodos remotos. O
* comportamento destes métodos, são desenvolvido
* no programa IEEE1451_NCABlock.java.
*/

//Conjunto de classes requeridas para compilar o programa
import java.rmi.*; // classe que possibilita a o acesso remoto.

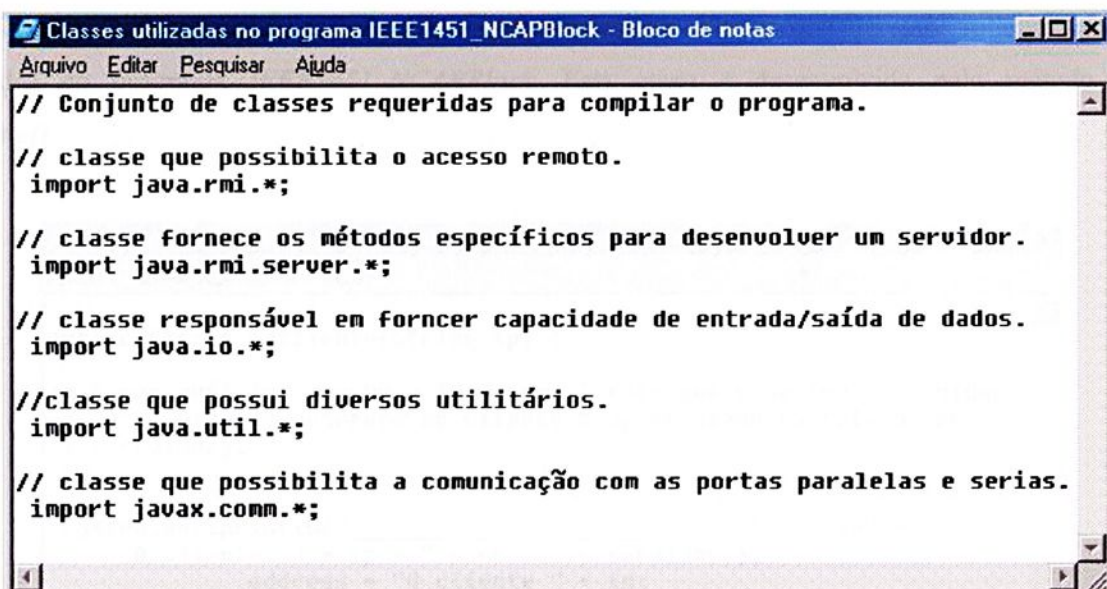
// definição da assinatura da interface remota(IEEE1451_BaseTransducerBlock)
public interface IEEE1451_BaseTransducerBlock extends Remote {

// definição dos métodos remotos
    public String IEEE1451NCAP(String dado) throws RemoteException;
    public String IdCliente(String ip) throws RemoteException;
}
```

Figura 6.4: Código fonte da interface remota *IEEE1451_BaseTransducerBlock*.

Nas Figuras 6.5 a 6.10 apresentam-se, etapas do código fonte do programa *IEEE1451_NCABlock*.

Pode-se observar na Figura 6.5, quais as classes que fornecem métodos e variáveis para compor o programa *IEEE1451_NCABlock*.



```
// Conjunto de classes requeridas para compilar o programa.

// classe que possibilita o acesso remoto.
import java.rmi.*;

// classe fornece os métodos específicos para desenvolver um servidor.
import java.rmi.server.*;

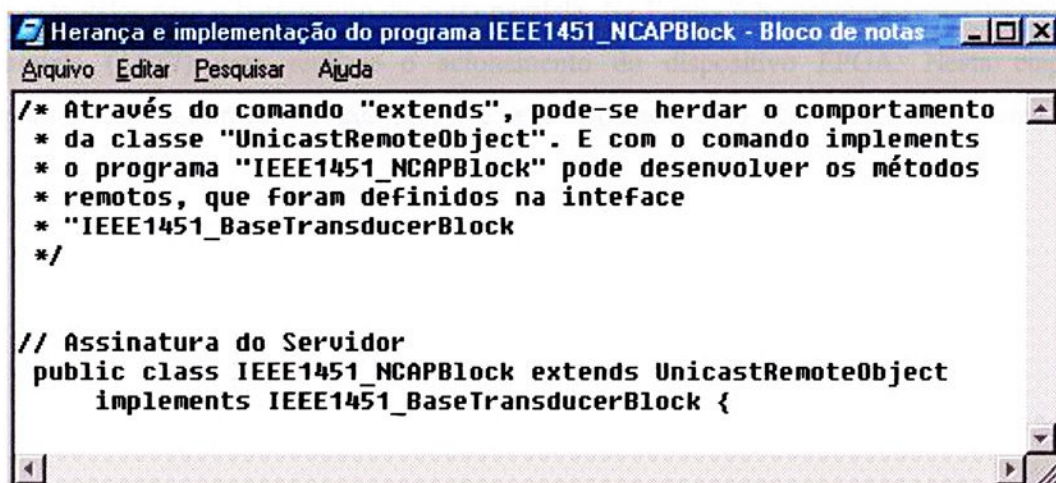
// classe responsável em fornecer capacidade de entrada/saída de dados.
import java.io.*;

// classe que possui diversos utilitários.
import java.util.*;

// classe que possibilita a comunicação com as portas paralelas e serias.
import javax.comm.*;
```

Figura 6.5: Classes utilizadas no programa *IEEE1451_NCAPBlock*.

Na Figura 6.6, mostra-se a linha de código, responsável por proporcionar ao programa *IEEE1451_NCAPBlock*, capacidade de herdar o comportamento da classe *UnicastRemoteObject* e implementar os métodos da interface *IEEE1451_BaseTransducerBlock*

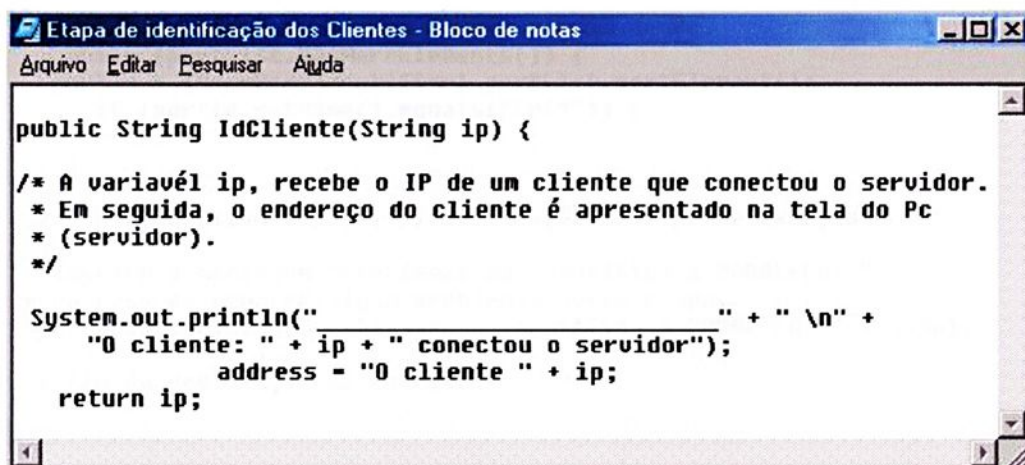


```
/* Através do comando "extends", pode-se herdar o comportamento
 * da classe "UnicastRemoteObject". E com o comando implements
 * o programa "IEEE1451_NCAPBlock" pode desenvolver os métodos
 * remotos, que foram definidos na interface
 * "IEEE1451_BaseTransducerBlock
 */

// Assinatura do Servidor
public class IEEE1451_NCAPBlock extends UnicastRemoteObject
    implements IEEE1451_BaseTransducerBlock {
```

Figura 6.6: Herança e implementação do programa *IEEE1451_NCAPBlock*.

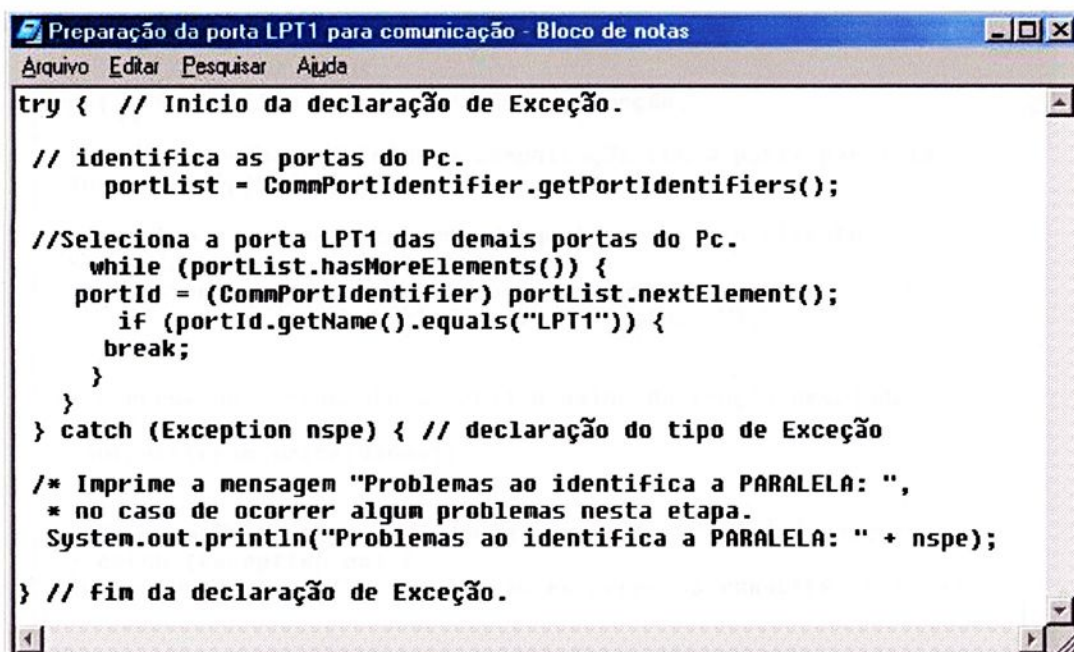
Na Figura 6.7, mostra-se etapa de identificação dos clientes que desejam solicitar as atividades do programa *IEEE1451_NCAPBlock*. Esta etapa é desenvolvida pelo método *IdCliente()*.



```
public String IdCliente(String ip) {  
    /* A variável ip, recebe o IP de um cliente que conectou o servidor.  
     * Em seguida, o endereço do cliente é apresentado na tela do Pc  
     * (servidor).  
     */  
    System.out.println("_____ " + " \n" +  
        "0 cliente: " + ip + " conectou o servidor");  
        address = "0 cliente " + ip;  
    return ip;  
}
```

Figura 6.7: Etapa de identificação dos clientes.

Nas Figuras 6.8 e 6.9 descrevem-se características do comportamento do método *IEEE1451NCAP()*, que tem como finalidade receber o valor da função enviada pelo cliente e transferir este valor para o barramento da porta paralela. Na Figura 6.8 apresenta-se a seleção da porta paralela (*LPT1*) para realizar o acionamento do dispositivo FPGA. Nesta etapa, implementa-se uma declaração de exceção, que será disparada, caso ocorra algum problema na identificação da *LPT1*.



```
try { // Início da declaração de Exceção.

    // identifica as portas do Pc.
    portList = CommPortIdentifier.getPortIdentifiers();

    //Seleciona a porta LPT1 das demais portas do Pc.
    while (portList.hasMoreElements()) {
        portId = (CommPortIdentifier) portList.nextElement();
        if (portId.getName().equals("LPT1")) {
            break;
        }
    }
} catch (Exception nspe) { // declaração do tipo de Exceção

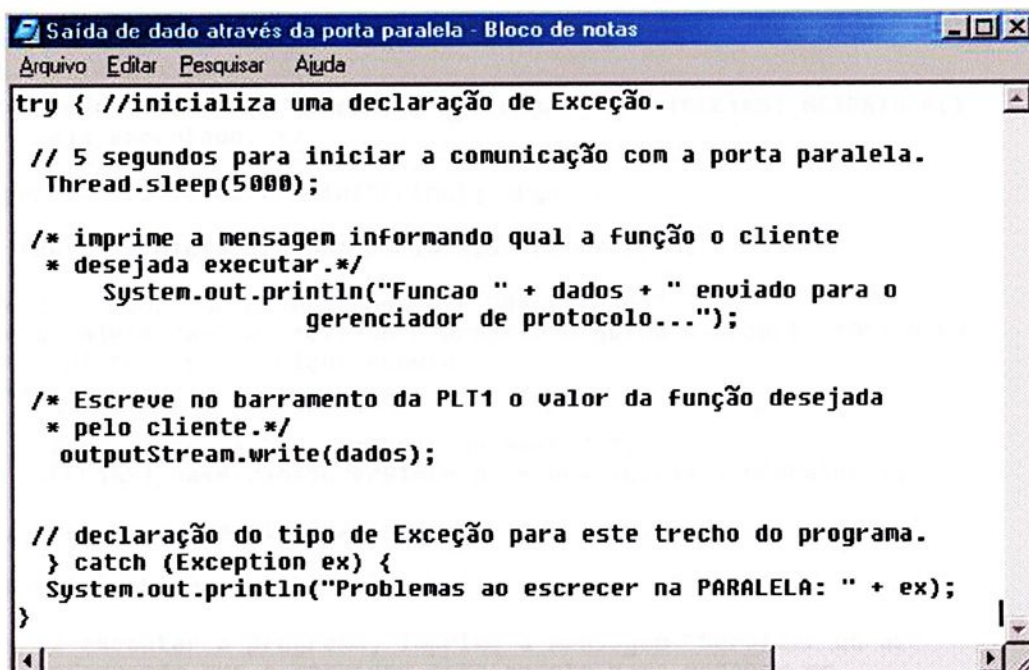
    /* Imprime a mensagem "Problemas ao identifica a PARALELA: ",
     * no caso de ocorrer algum problemas nesta etapa.
     */
    System.out.println("Problemas ao identifica a PARALELA: " + nspe);
} // fim da declaração de Exceção.
```

Figura 6.8: Seleciona-se a porta LPT1 para realizar a comunicação.

Nota-se na Figura 6.9, como o programa *IEEE1451_NCAPBlock* escreve um valor no barramento da *LPT1*. Ao enviar um endereço de função (1 byte D0 – D7) para o barramento da *LPT1*, o dado estará disponível após 5 segundos (este tempo é opcional³⁹). Para escrever este endereço no barramento, o objeto da classe *OutputStream*, insere no método *write()* a função desejada. Nesta etapa, conforme Figura 6.9, pode-se observar como a declaração de uma exceção, analisa o funcionamento de um trecho específico do programa.

³⁹ O tempo de 5 segundo foi para analisar a capacidade de temporização da tecnologia Java, porém, este recurso não é essencial para a comunicação com o Gerenciador de Protocolo.





```
try { //inicializa uma declaração de Exceção.

    // 5 segundos para iniciar a comunicação com a porta paralela.
    Thread.sleep(5000);

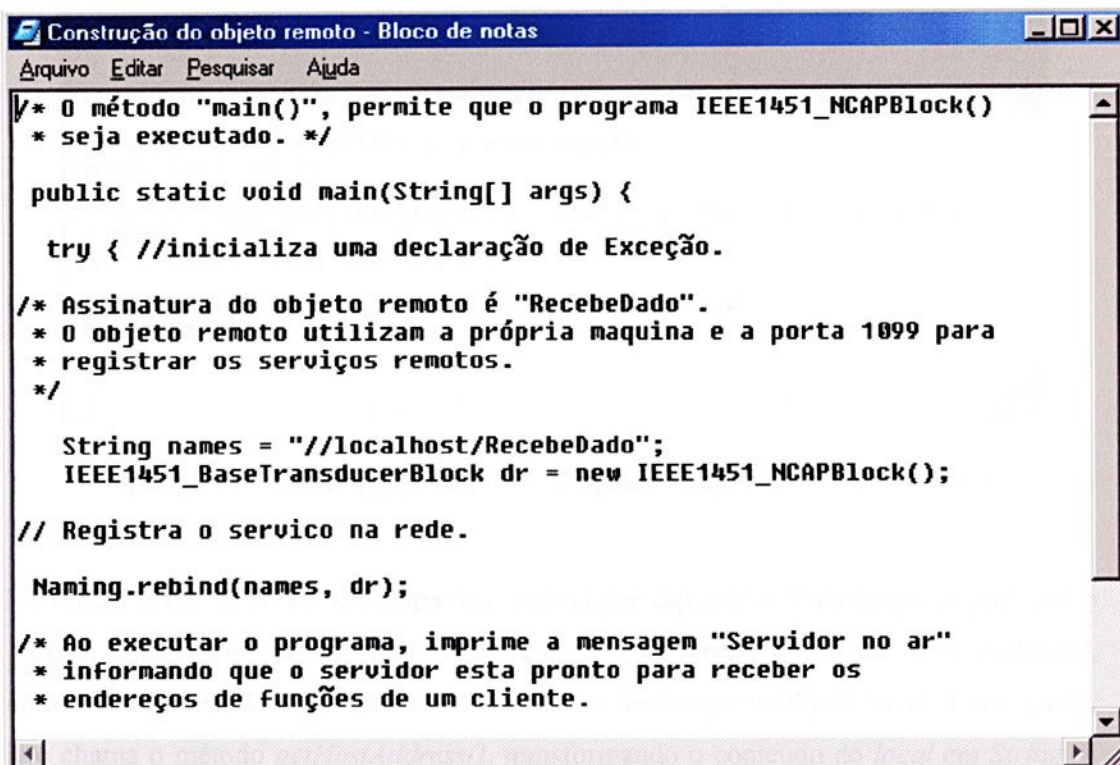
    /* imprime a mensagem informando qual a função o cliente
     * desejada executar.*/
    System.out.println("Funcao " + dados + " enviado para o
                        gerenciador de protocolo...");

    /* Escreve no barramento da PLT1 o valor da função desejada
     * pelo cliente.*/
    outputStream.write(dados);

    // declaração do tipo de Exceção para este trecho do programa.
    } catch (Exception ex) {
        System.out.println("Problemas ao escrever na PARALELA: " + ex);
    }
}
```

Figura 6.9: O endereço de função é enviado para LPT1.

Na Figura 6.10 demonstra-se a construção do objeto remoto. Este objeto foi registrado na rede da seguinte forma, *Naming.rebind(names, dr)*, sendo desenvolvido dentro do método *main()* do programa *IEEE1451_NCAPBlock*. A variável *names* representa o local que o servidor está sendo executado, e o objeto *dr*, direciona os métodos remotos para serem utilizados no programa.



```
Construção do objeto remoto - Bloco de notas
Arquivo  Editar  Pesquisar  Ajuda

/* O método "main()", permite que o programa IEEE1451_NCABlock()
 * seja executado. */

public static void main(String[] args) {

    try { //inicializa uma declaração de Exceção.

/* Assinatura do objeto remoto é "RecebeDado".
 * O objeto remoto utilizam a própria maquina e a porta 1099 para
 * registrar os serviços remotos.
 */

        String names = "//localhost/RecebeDado";
        IEEE1451_BaseTransducerBlock dr = new IEEE1451_NCABlock();

// Registra o servico na rede.

        Naming.rebind(names, dr);

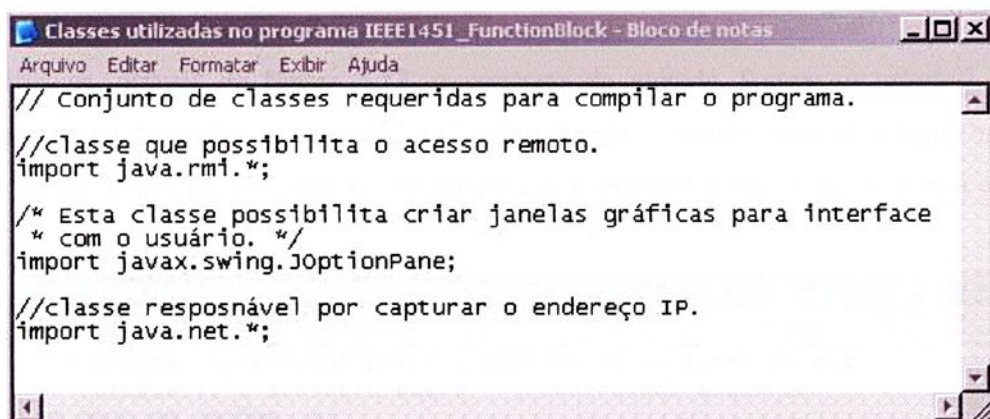
/* Ao executar o programa, imprime a mensagem "Servidor no ar"
 * informando que o servidor esta pronto para receber os
 * endereços de funções de um cliente.
 */

    }
```

Figura 6.10: Construção do objeto remoto.

Nas Figuras 6.11, 6.12, 6.13, 6.14 e 6.15 exibe-se parte do código fonte do programa *IEEE1451_FunctionBlock*.

Na Figura 6.11 descrevem-se quais as classes, que fornecem métodos e variáveis para compor o programa *IEEE1451_FunctionBlock*.



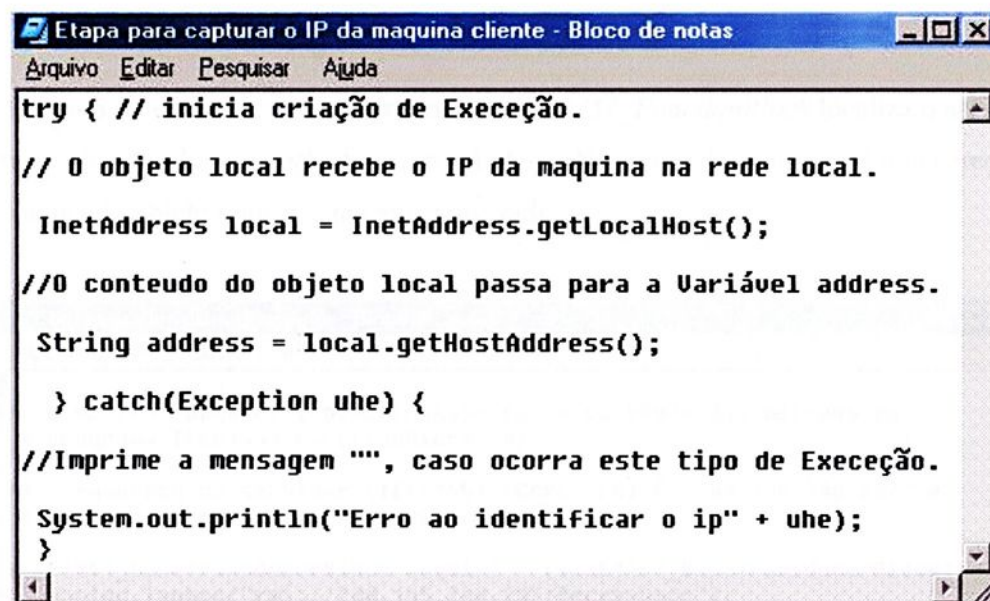
```
// Conjunto de classes requeridas para compilar o programa.
// classe que possibilita o acesso remoto.
import java.rmi.*;

/* Esta classe possibilita criar janelas gráficas para interface
 * com o usuário. */
import javax.swing.JOptionPane;

// classe responsável por capturar o endereço IP.
import java.net.*;
```

Figura 6.11: Classes para compilar o programa *IEEE1451_FunctionBlock*.

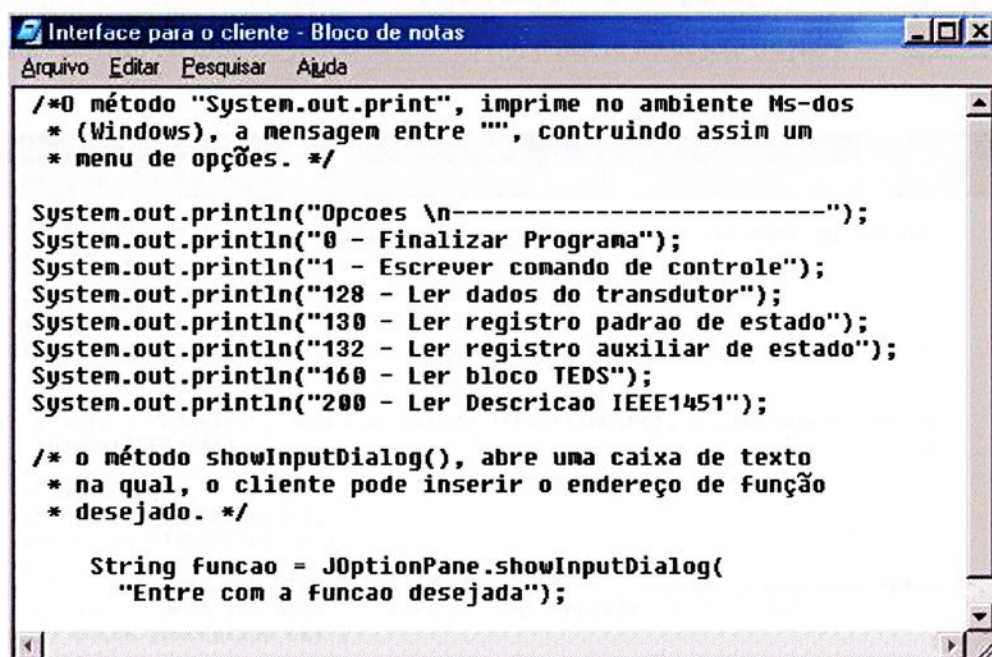
Na Figura 6.12, descreve-se a etapa responsável por capturar o IP da máquina, para que o cliente possa identificar-se no servidor. Para realizar esta tarefa, o método *getLocalHost()*, pertencente a classe *InetAddress*, captura o IP e insere o endereço no objeto *local*. Em seguida, este objeto chama o método *getHostAddress()*, transformando o conteúdo do *local* em *String*. E este valor *String*, é representado pela variável *address*. Portanto, o cliente envia conteúdo da variável *address*, para identificar-se no servidor.



```
try { // inicia criação de Exceção.
// O objeto local recebe o IP da maquina na rede local.
    InetAddress local = InetAddress.getLocalHost();
//O conteudo do objeto local passa para a Variável address.
    String address = local.getHostAddress();
} catch(Exception uhe) {
//Imprime a mensagem "", caso ocorra este tipo de Exceção.
    System.out.println("Erro ao identificar o ip" + uhe);
}
```

Figura 6.12: Método *enviaIp()* captura endereço IP da máquina.

Verifica-se, através da Figura 6.13, o emprego do método *System.out.println()*, para implementar o menu de opções e com o código "*String funcao = JOptionPane.showInputDialog("Entre com a função desejada")*", cria-se um campo que possibilita o cliente entrar com a opção desejada.



```

/*0 método "System.out.print", imprime no ambiente Ms-dos
 * (Windows), a mensagem entre "", contruindo assim um
 * menu de opções. */

System.out.println("Opcoes \n-----");
System.out.println("0 - Finalizar Programa");
System.out.println("1 - Escrever comando de controle");
System.out.println("128 - Ler dados do transdutor");
System.out.println("130 - Ler registro padrao de estado");
System.out.println("132 - Ler registro auxiliar de estado");
System.out.println("160 - Ler bloco TEDS");
System.out.println("200 - Ler Descricao IEEE1451");

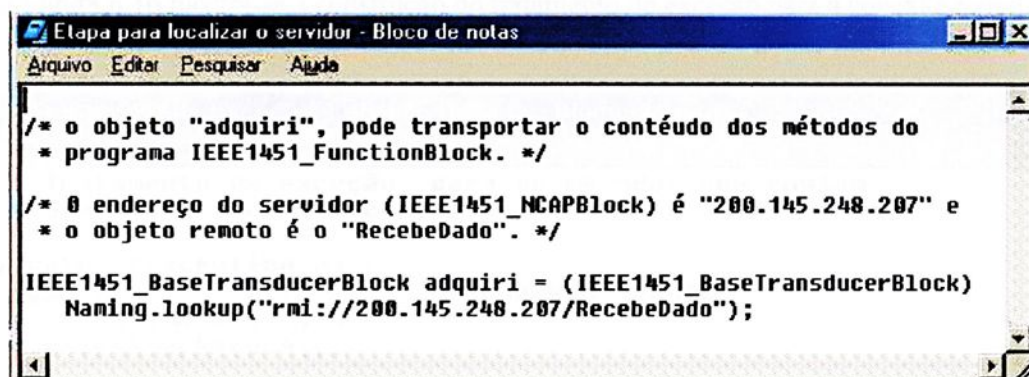
/* o método showInputDialog(), abre uma caixa de texto
 * na qual, o cliente pode inserir o endereço de função
 * desejado. */

String funcao = JOptionPane.showInputDialog(
    "Entre com a funcao desejada");

```

Figura 6.13: Interface para o cliente.

Na Figura 6.14 exibe-se, como o programa *IEEE1451_FunctionBlock* localiza o servidor. Ou seja, através do método *lookup()* pertencente á classe *Naming*, informa-se qual o endereço do servidor e o nome do objeto remoto que vai ser utilizado.



```

/* o objeto "adquiri", pode transportar o conteúdo dos métodos do
 * programa IEEE1451_FunctionBlock. */

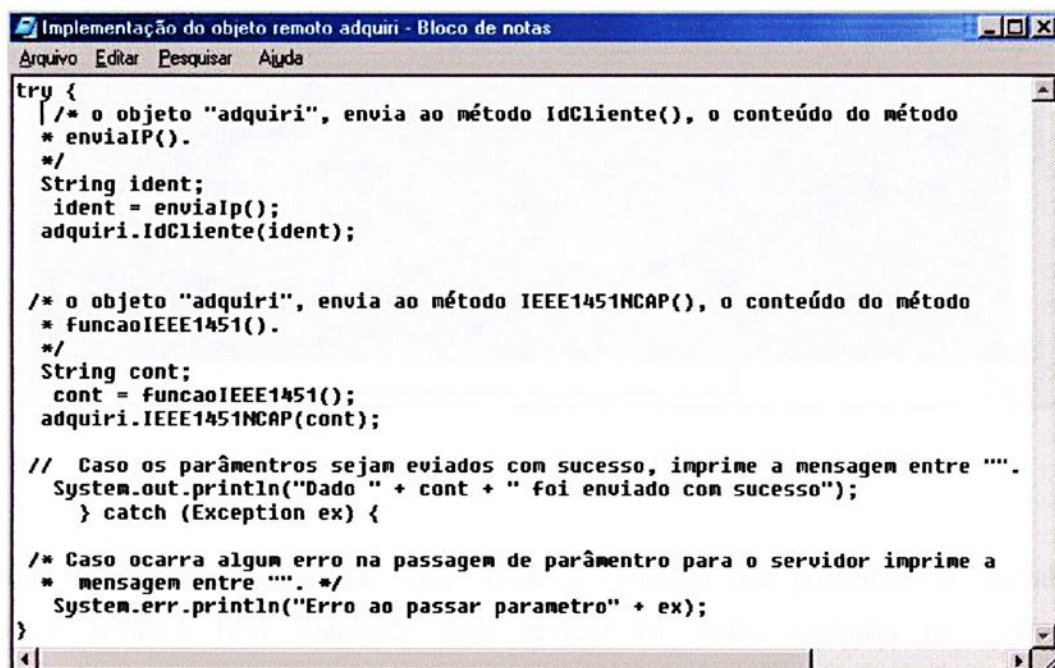
/* 0 endereço do servidor (IEEE1451_NCAPBlock) é "200.145.248.207" e
 * o objeto remoto é o "RecebeDado". */

IEEE1451_BaseTransducerBlock adquiri = (IEEE1451_BaseTransducerBlock)
    Naming.lookup("rmi://200.145.248.207/RecebeDado");

```

Figura 6.14: Etapa para localizar o servidor..

Descreve-se na Figura 6.15 a formação do conteúdo que será enviado para o servidor, ou seja, ao programa *IEEE1451_NCAPBlock*. A variável *ident* recebe o parâmetro (tipo *String*) do método *enviaIp()*, e insere este parâmetro no objeto *adquiri*. Em seguida, a variável *cont* recebe o parâmetro (tipo *String*) do método *funcaoIEEE1451()*, e insere no objeto *adquiri*.



```
try {
    /* o objeto "adquiri", envia ao método IdCliente(), o conteúdo do método
    * enviaIP().
    */
    String ident;
    ident = enviaIp();
    adquiri.IdCliente(ident);

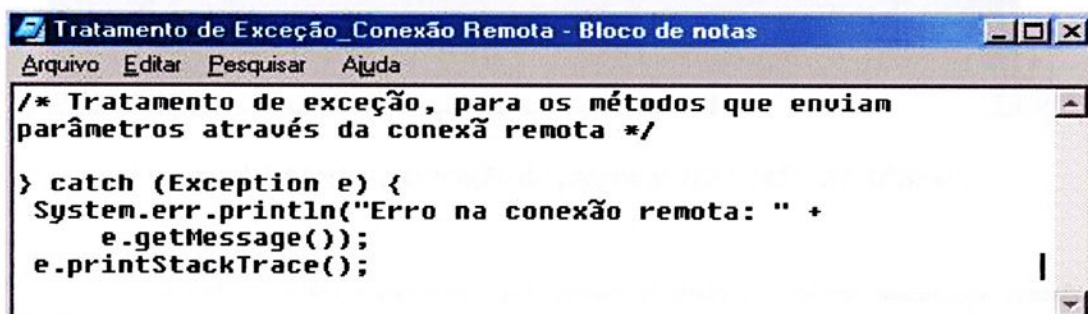
    /* o objeto "adquiri", envia ao método IEEE1451NCAP(), o conteúdo do método
    * funcaoIEEE1451().
    */
    String cont;
    cont = funcaoIEEE1451();
    adquiri.IEEE1451NCAP(cont);

    // Caso os parâmetros sejam enviados com sucesso, imprime a mensagem entre "".
    System.out.println("Dado " + cont + " foi enviado com sucesso");
} catch (Exception ex) {

    /* Caso ocorra algum erro na passagem de parâmetro para o servidor imprime a
    * mensagem entre "". */
    System.err.println("Erro ao passar parametro" + ex);
}
```

Figura 6.15: Implementação do objeto remoto "adquiri".

Na Figura 6.16 mostra-se a construção do tratamento de exceção para a conexão remota.



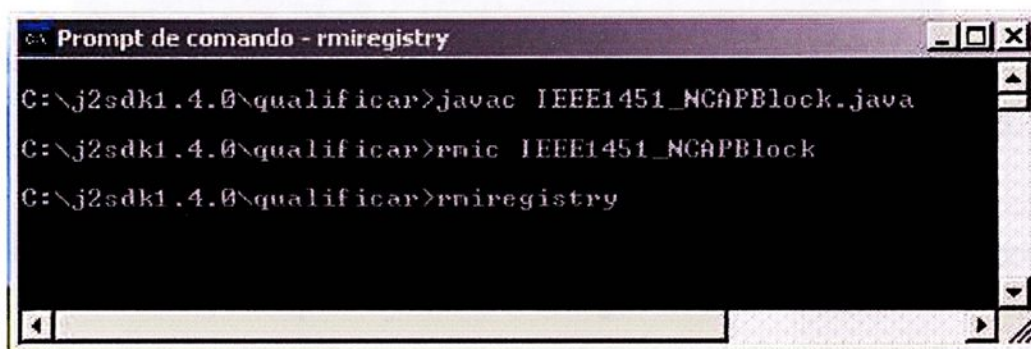
```
/* Tratamento de exceção, para os métodos que enviam
parâmetros através da conexão remota */

} catch (Exception e) {
    System.err.println("Erro na conexão remota: " +
        e.getMessage());
    e.printStackTrace();
}
```


Figura 6.16: Tratamento de Exceção para conexão remota.

Na Figura 6.17 mostra-se os comandos necessários para compilar o programa *IEEE1451_NCABlock* (*javac* e *rmic*) e registrar o objeto remoto (*rmiregistry*) localizado na máquina local⁴⁰.

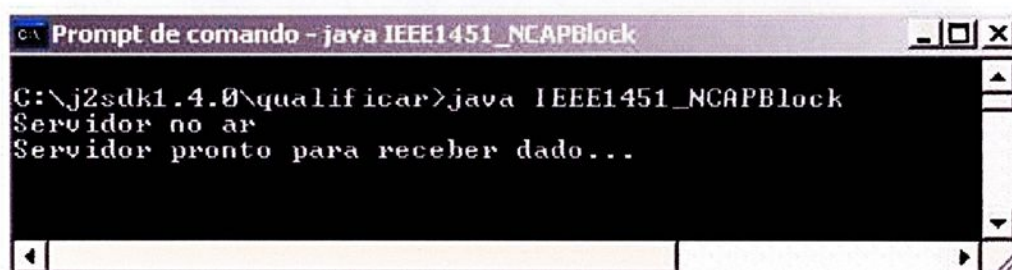
O programa *IEEE1451_BaseTransducerBlock* será compilado automaticamente com o *IEEE1451_NCABlock*.



```
CA Prompt de comando - rmiregistry
C:\j2sdk1.4.0\qualificar>javac IEEE1451_NCABlock.java
C:\j2sdk1.4.0\qualificar>rmic IEEE1451_NCABlock
C:\j2sdk1.4.0\qualificar>rmiregistry
```

Figura 6.17: Etapas para compilar e registrar o *IEEE1451_NCABlock*.

Pode-se observar, através da Figura 6.18, o comando que possibilita ao servidor *IEEE1451_NCABlock* ficar preparado para receber os dados enviados pelo cliente *IEEE1451_FunctionBlock*.



```
CA Prompt de comando - java IEEE1451_NCABlock
C:\j2sdk1.4.0\qualificar>java IEEE1451_NCABlock
Servidor no ar
Servidor pronto para receber dado...
```

Figura 6.18: Mostra a execução do programa *IEEE1451_NCABlock*.

⁴⁰ Máquina local, o objeto remoto implementado pelo servidor encontra-se na mesma máquina que executa

A compilação e a execução do programa *IEEE1451_FunctionBlock* estão ilustradas na Figura 6.19.



Figura 6.19: Etapas para compilar e executar o *IEEE1451_FunctionBlock*.

O servidor identifica um determinado cliente imediatamente após a execução do programa *IEEE1451_FunctionBlock*, como pode ser verificado na Figura 6.20.

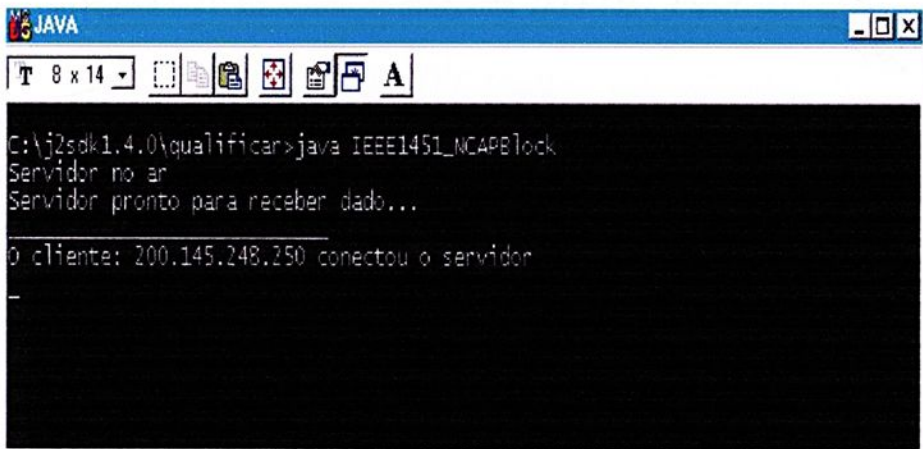


Figura 6.20: Servidor identifica um cliente.

Na Figura 6.21 descreve-se os endereços de funções que um cliente pode enviar para o Grenciador de Potocolo exercer suas atividades, além de ilustrar a interface gráfica utilizada para

o programa.

o usuário inserir a função desejada. Estes endereços de funções são definidos no padrão IEEE1451.2, porém, podem ser construídos conforme a determinação do projetista.

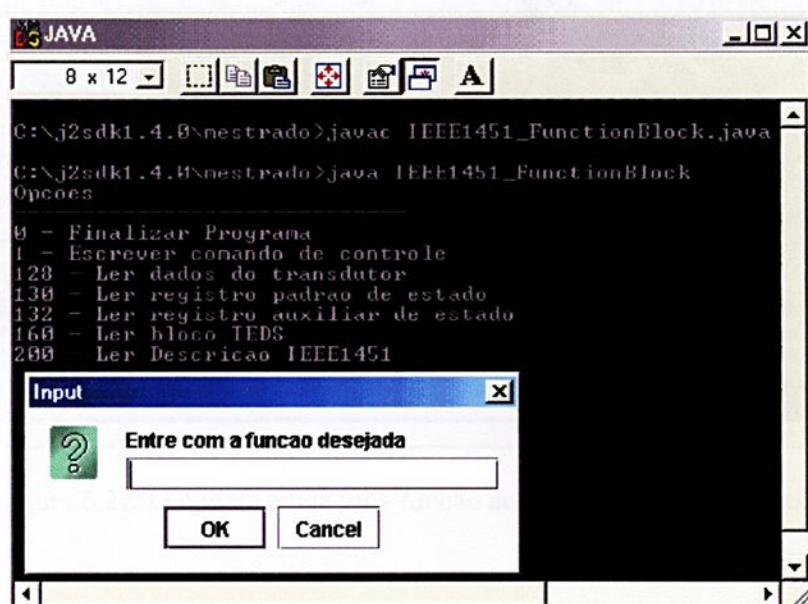


Figura 6.21: Execução do programa *IEEE1451_FunctionBlock*.

Na Figura 6.22 exibe-se a escolha da função desejada pelo cliente “160”. Assim ao *clicar* no botão de [ok], este valor é enviado ao servidor.

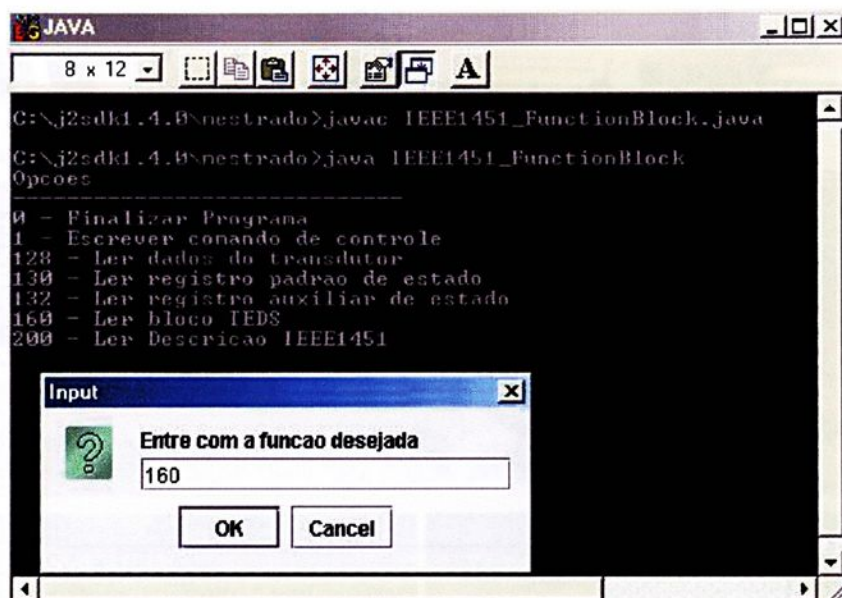


Figura 6.22: O cliente envia uma função ao gerenciador de protocolo.

Pode-se observar, na Figura 6.23, um dispositivo⁴¹ implementando exatamente o valor “160”, que foi enviado remotamente por um cliente utilizando o software NCAP, ou seja, evidencia-se que o dispositivo é acionado com sucesso.

⁴¹ O dispositivo empregado para testar o software NCAP é um protoboard que contém um conjunto de Leds.

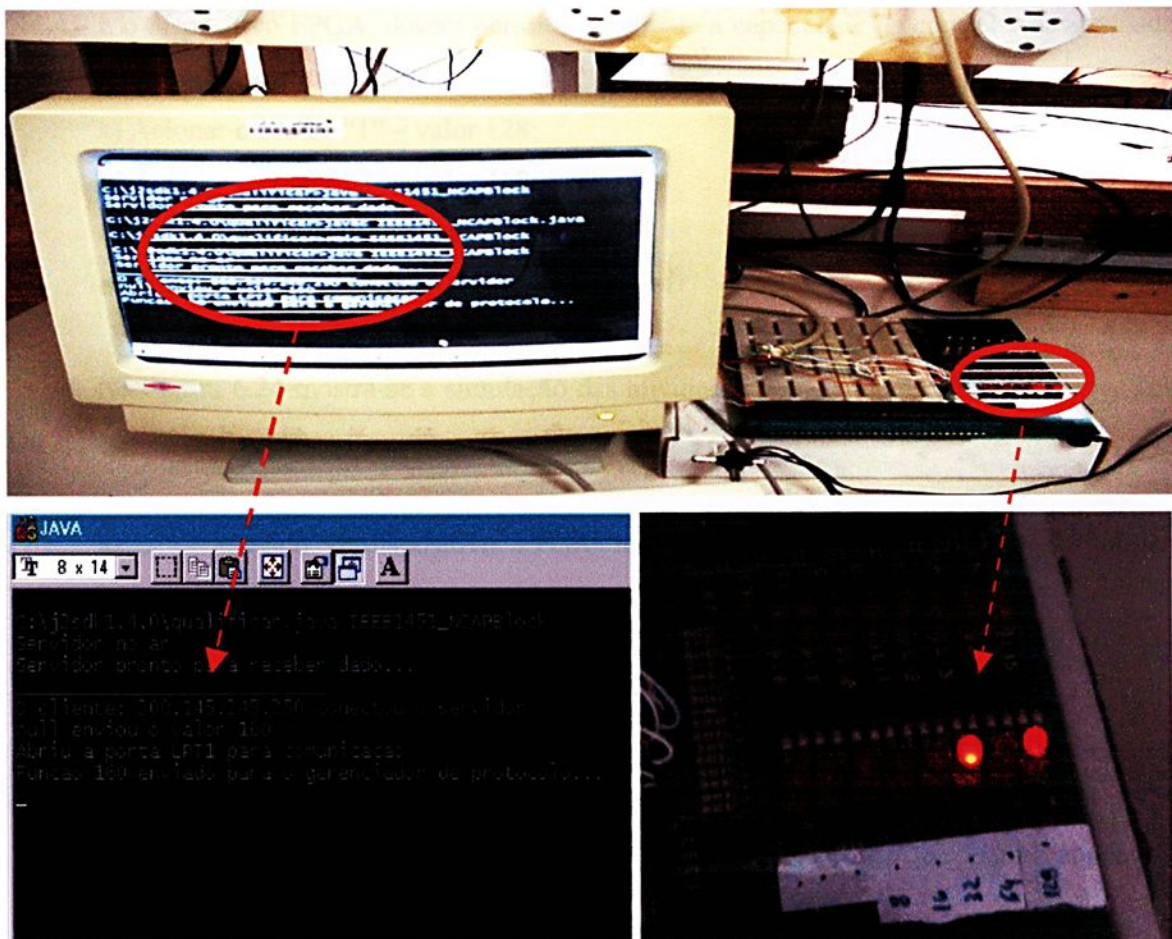


Figura 6.23: O servidor transfere o valor “160” ao dispositivo.

O dispositivo acionado pelo servidor, mostra que o software NCAP, possui capacidade de enviar um valor com sucesso para o Gerenciador de Protocolo.

Outro teste, realizado com o software, foi o controle das atividades de dois *coolers*. Neste teste, o software envia um endereço de função para um dispositivo FPGA, e este executa uma lógica de endereçamento, para controlar as atividades dos *cooler*, exercendo, desta forma, a mesma função do Gerenciador de Protocolo. Para facilitar o entendimento deste teste, foram alterados os nomes dos três programas, ou seja, o programa *IEEE1451_BaseTransducerBlock* foi denominado de *Cooler*, o programa *IEEE1451_NCAPBlock* foi definido como *AcionaCooler* e, finalmente, o programa *IEEE1451_FunctionBlock* foi nomeado de *AcionadorCooler*. O software

NCAP e o dispositivo FPGA, devem permitir ao cliente a capacidade de controlar as atividades dos cooler, com as seguintes opções:

- a) Acionar o cooler “1” – valor 128;
- b) Acionar o cooler “2” – valor 160;
- c) Acionar os dois coolers – valor 130;
- d) Desligar os dois coolers – 1.

Na Figura 6.24 mostra-se a simulação das atividades que o módulo FPGA deve realizar. Ou seja:

- a) quando o dispositivo FPGA recebe o valor “160”, a saída do cooler1 permanece no estado atual e a saída do cooler2 vai para “1”;
- b) quando o dispositivo FPGA recebe o valor “130”, a saída do cooler1 vai para “1” e a saída do cooler2 vai para “1”;
- c) quando o dispositivo FPGA recebe o valor “128”, a saída do cooler1 vai para “1” e a saída do cooler2 permanece no estado atual;
- d) quando o dispositivo FPGA recebe o valor “1”, a saída do cooler1 vai para “0” e a saída do cooler2 vai para “0”.

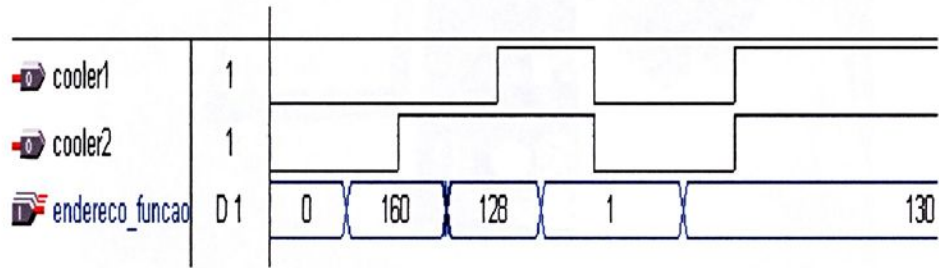


Figura 6.24: Simulação da lógica de endereçamento.

Na Figura 6.25 exibe-se o instante em que o servidor (*AcionaCooler*), detecta qual cliente está solicitando seus serviços e o valor que deve ser enviado para o módulo FPGA.

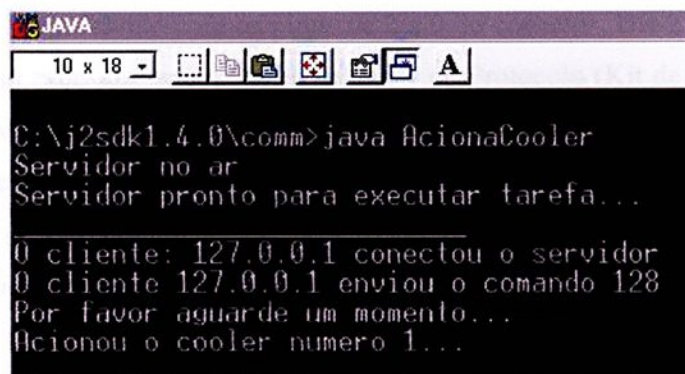


Figura 6.25: Execução do *AcionaCooler*.

Na Figura 6.26 ilustra-se o protótipo utilizado para controlar as atividades dos *coolers*.

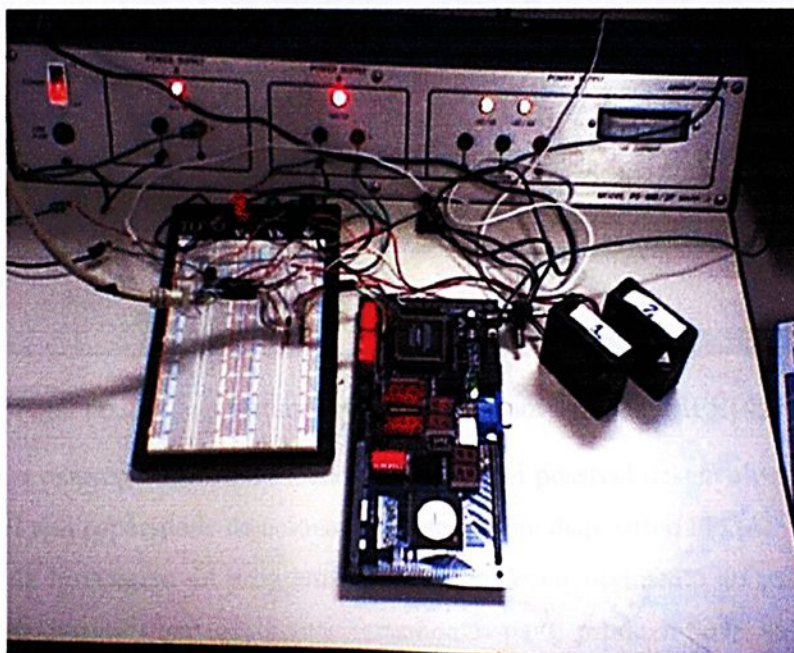


Figura 6.26: Arquitetura para testar o software.

A arquitetura para acionar um dispositivo (servidor), em conformidade com o padrão IEEE1451 é apresentada na Figura 6.27, ou seja, o software NCAP acionando um transdutor, com o auxílio do Gerenciador de Protocolo e do STIM. Nesta arquitetura, o cabo que sai da porta paralela do PC é conectado em um *protoboard*, para criar um barramento de dados. Este

barramento de dados é desenvolvido com os pinos de 2 a 9 do conector DB-25 (observação: os pinos de 18 a 25, devem estar aterrados). O Gerenciador de Protocolo (Kit de desenvolvimento contendo um dispositivo FPGA Acex EP1K50TC144-3, da Altera) recebe os endereços de funções através do barramento de dados. Em seguida, o Gerenciado de Protocolo envia a instrução para o STIM (Kit de desenvolvimento UP1⁴²) e finalmente o dispositivo conectado ao STIM, executa a atividade requerida pelo cliente.

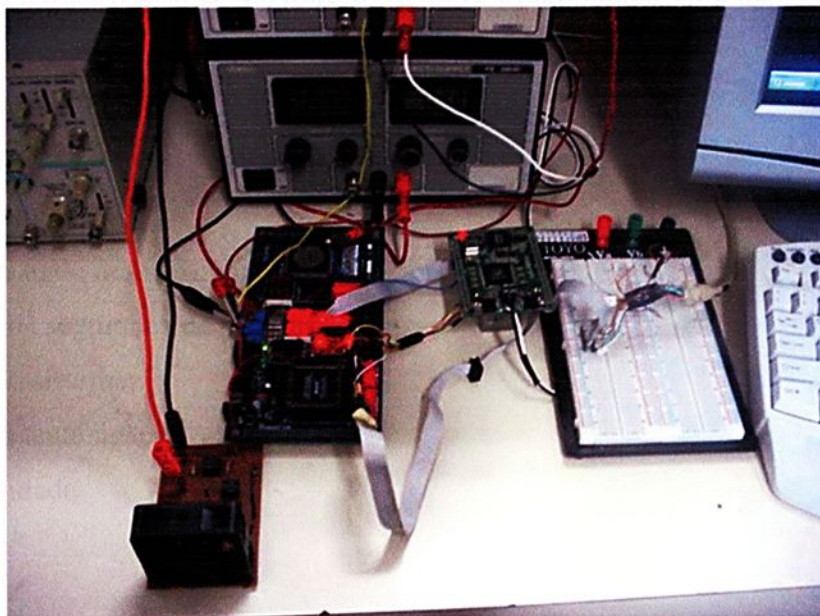


Figura 6.27: Arquitetura conformidade com o padrão IEEE1451.

Assim, com o emprego apenas da tecnologia Java, foi possível desenvolver um software para o NCAP, o qual tem capacidade de acionar via Internet um dispositivo FPGA.

O emprego da linguagem de programação Java no desenvolvimento do software para o NCAP, apresenta-se como uma excelente ferramenta para produzir um software livre, terminologia que refere-se ao grau de liberdade que um programa deve oferecer ao usuário [43]:

- a) liberdade de executar o programa, para qualquer protótipo;
- b) liberdade de analisar o código fonte e adaptá-lo para as necessidades de cada usuário;
- c) liberdade de redistribuir cópias;

⁴² Placa do Programa Universitário da ALTERA.

- d) liberdade de aperfeiçoar o programa e disponibilizar os aperfeiçoamentos, de modo que toda a comunidade se beneficie.

O acesso ao código fonte é fundamental para que um programa seja considerado software livre. Seguindo esta tendência, diversas corporações como a *IBM*, *Sun Microsystems*, *HP*, *RedHat*, *Netscape*, *Governo Federal brasileiro*, etc. já assumiram publicamente o interesse em adotar o sistema de software livres. O software livre, deve proporcionar para indústria vantagens como:

- a) redução de custos;
- b) aumento da qualidade;
- c) redução do prazo de desenvolvimento;
- d) comprovação da segurança de programas;
- e) longevidade do sistema;
- f) facilidade na manutenção;
- g) maior popularidade.

6.3 Trabalhos Futuros

As tecnologias disponíveis atualmente possibilitam o desenvolvimento de diversos modelos do padrão IEEE1451. Nesta seção são propostas algumas ferramentas para aprimorar o projeto iniciado neste trabalho.

6.3.1 Porta USB



A porta USB (Universal Serial Bus) é uma tendência atual e está presente nos PCs mais modernos. Desta forma, a porta USB pode ser utilizada para acionar o gerenciador de protocolo proporcionando vantagem como, maior velocidade na troca de informações.

6.3.2 Jini

A Sun Microsystems possui um sistema que possibilita a integração de dispositivos eletrônicos. Este sistema, denominado de Jini, é uma tecnologia que tem como objetivo aceitar diferentes protocolos e, portanto, constitui uma interessante ferramenta para desenvolver o software NCAP.

6.3.3 CORBA

O sistema CORBA (*Common Object Request Broker Architecture*) pode ser empregado para realizar o monitoramento remoto via Internet. Este sistema supera a tecnologia RMI, por aceitar diversas linguagens de programação e diferentes protocolos de rede.

6.3.4 Processador Java

O NCAP pode ser executado em um microprocessador Java como, por exemplo, o *picoJava*, que inclui a JVM. A Sun desenvolveu uma série de microprocessadores (*picoJava*, *microJava* e *ultraJava*) que possibilitam a construção de sistemas embarcados e aumentam a velocidade na execução das aplicações.



6.3.5 Interface gráfica

A construção de interface gráfica torna o NCAP uma aplicação mais amigável. A linguagem Java possui muitos recursos para desenvolver interfaces gráficas.

6.3.6 Applet

O emprego de *applets* no nível superior do software NCAP, facilita a implementação da aplicação via Internet.

6.3.7 Cliente via celular

Como já existe API que desenvolve aplicações para rodar em celular, pode-se implementar um serviço orientado a clientes. Assim, estes clientes podem acessar o NCAP via telefone celular.

6.3.8 NCAP Java x C++

É possível desenvolver um NCAP utilizando C++ e comparar as vantagens e desvantagens entre Java e C++.

6.3.9 Can



É possível utilizar a rede CAN (*Controller Area Network*) no lugar da rede Ethernet e comparar as vantagens e desvantagens de cada rede.

6.3.10 Java 3D e Java Advance

É possível incrementar o software NCAP com interface gráfica 3D e inserir capacidade de capturar imagens, de forma que o cliente tenha uma visão virtual das atividades dos transdutores.

6.3.11 Banco de Dados

Construir um banco de dados no servidor, contendo informações das atividades requeridas por cada cliente. Desta forma, o servidor pode ter um controle no acesso ao dispositivo.

6.3.12 Desenvolver API Java

Desenvolver uma API Java para trabalhar no modo bidirecional da porta paralela, pois com o pacote Java COMM não foi possível realizar este processo.



7. Conclusão

Neste trabalho implementou-se um processador com capacidade de operar em rede de comunicação (NCAP) em conformidade com padrão IEEE1451.1. Desta forma, o trabalho envolve conceitos de rede de comunicação (Ethernet), de linguagem orientada a objeto (Java) e do padrão IEEE1451.

Para implementar o NCAP empregou-se, basicamente, um PC e a tecnologia Java. O PC implementou as partes físicas do NCAP (processador e o driver de rede) e possibilitou o controle do Gerenciador de Protocolo através da porta paralela.

A linguagem de programação Java foi aplicada para desenvolver as funções lógicas de um NCAP, dando origem ao software NCAP. Este software é capaz de proporcionar o controle via Internet, um dispositivo (Gerenciador de Protocolo) conectado na porta paralela de PC, baseando-se no modelo de comunicação cliente/servidor e independente do sistema operacional.

O emprego da API Java Comm, para possibilitar o controle de um dispositivo conectado na porta paralela de um PC, é uma contribuição expressiva desta pesquisa, uma vez, que não foram encontradas na literatura aplicações que comprovassem a potencialidade deste importante pacote Java.

A utilização da rede Ethernet no trabalho deveu-se à expansão de forma vertical dessa rede na automação atual, além de ser uma rede de baixo custo, padronizada e com possibilidade de se tornar interoperável, ou seja, que possui características fundamentais para desenvolver a comunicação conforme determina o padrão IEEE1451.1.

O software NCAP possibilita o acionamento de um dispositivo conectado na porta paralela de um microcomputador. Testes realizados comprovaram sua eficiência no acionamento de um conjunto de *leds*. Os testes também comprovaram que o software é eficiente no acionamento de um dispositivo FPGA, como foi verificado no controle das atividades de dois *coolers*.



Como produto final, o grupo de pesquisa responsável pela implementação deste projeto, pretende tornar a parte lógica do NCAP, um software livre, com o objetivo de proporcionar aos projetos de Instrumentação Distribuída maior diversidade nas aplicações industriais e que torna-se uma ferramenta acessível ao consumidor. Assim, as tecnologias que este trabalho utilizou, para construir o NCAP, proporcionaram ao sistema de monitoramento remoto maior flexibilidade na manutenção, redução de custo e maior integração de dispositivos inteligentes.



REFERÊNCIAS

- [1] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. IEEE 1451 website. What is IEEE 1451?. Disponível em: <http://ieee1451.nist.gov>>. Acesso em: 15 dez 2003.
- [2] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Other 1451 website. Disponível em: <http://ieee1451.nist.gov/>>. Acesso em: 28 dez 2003
- [3] JOHNSON, R. N. IEEE standard 1451.0 defining the core features of smart sensor to facilitate broader adoption. Sensor Expo 2003, Chigago, 3 jun. 2003, Disponível em: < <http://www.telemonitor.com/download.html> >. Acesso em: 5 jan. 2004
- [4] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Instrumentation and Measurement Society. **IEEE standard for a smart transducer interface for sensor and actuators network capable application processor (NCAP) information model**. New York: IEEE, 1999. 349p.
- [5] WOODS, S P.; BRYZEK, J.; CHEN, S.; CRANMER, J.; EL-KAREH, E. V.; GEIPEL, M.; GEN-KUONG, F.; HOULDSWORTH, J.; LeCOMTE, N.; LEE, K.; MATTES, M. F.; RASMUSSEN, D. E. IEEE - P1451.2 smart transducer interface module. Disponível em: < <http://ieee1451.nist.gov> >. Acesso em: 20 dez. 2003.
- [6] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY.– 1451.3 website. Disponível em: <http://ieee1451.nist.gov/>>. Acesso em: 28 dez. 2003
- [7] TORBEN, R. The IEEE1451.4 proposed standard emerging compatible smart transducers and systems. Papers in PDF format, Proposed IEEE1451.4. Disponível em: < <http://ieee1451.nist.gov/> >. Acesso em: 25 jan. 2004.
- [8] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS P1451.5 PROJECT. Draft standard for a smart transducer interface for sensor and actuators wireless communication protocols and transducer Electronic Data Sheets. Instrumentation and Measurement Society. Disponível em: <http://grouper.ieee.org/groups/1451/5/>>. Acesso em: 18 jan. 2004.



- [9] BATISTA, E. A.; ROSSI, S. R.; SILVA, A. C. R.; CARVALHO, A. A. KITANO, C. **Proposta de implementação de um sistema de monitoramento remoto baseado no padrão IEEE1451.** In: World Congress Education Tcnology Engineer, WCETE, 19, 2004, Proced... Santos, art 538.

- [10] SCHNEEMAN, R. Implementing a standards-based distributed measurements and control application on the internet. Papers in PDF format, Implementing DMC. Gaithersburg, Maryland 1999. Disponível em: [<http://ieee1451.nist.gov/>](http://ieee1451.nist.gov/). Acesso em: 10 jan. 2004.

- [11] CAMARA, L.; RUIZ, O.; SAMITIER, J. Complete IEEE-1451 node, STIM and NCAP, implemented for a CAN network. In: INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE, IMTC, 17, 2000, Barcelona **Proceedings of the...**, Barcelona: IEEE, 2000. v. 2, p.541- 545.

- [12] SVEDA, M.; VRBA, R. Sensor networking. In: ENGINEERING OF COMPUTER BASED SYSTEMS, ECBS, 8, 2001. **Proceedings of the...** IEEE, 2001. p.262-268.

- [13] DE CASTRO, A.; CHAQUET, J. M.; Morejon, E.; Riesgo, T.; Uceda, J. A system-on-chip for smart sensors. In: INTERNATIONAL SYMPOSIUM INDUSTRIAL ELECTRONICS, ISIE, 2002. **Proceedings of the...**, IEEE, 2002. v. 2 , p. 595 –599.

- [14] WOBSCHALL, D. An implementation of IEEE 1451 NCAP for internet access of serial port-based. In: SENSORS FOR INDUSTRY CONFERENCE, 2, 2002. **Conference....**IEEE, 2002. p.157- 160.

- [15] RAMOS, H.; PEREIRA, M.; VIEGAS, V.; POSTOLACHEI, O.; GIRAO, P. A virtual instrument to test smart transducer interface modules (STIMs). In: INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE, IMTC, 3, 2003. **Proceedings of the...**, IEEE, 2003. v. 1, p.772 –775.

- [16] MEILIR, P-J. O que todo programador deveria saber sobre projeto orientado a objeto. Rio de Janeiro: Makron Books, 1997. 386p.

- [17] TANENBAUM, A. S. **Redes de computadores.** 3 ed. Rio de Janeiro: Campus, 1997. 884p.

- [18] MOREIRA. A. S. C. Redes Ethernet IEEE 802.3. Departamento de Engenharia Informática Instituto Superior de Engenharia do Porto. Porto, Portugal Disponível em: [<http://www.dei.isep.ipp.pt/~andre/documentos/ethernet.html>](http://www.dei.isep.ipp.pt/~andre/documentos/ethernet.html). Acesso em: 7 jan. 2004



- [19] MASSACHUSETTS INSTITUTE OF TECHNOLOGY, MIT SCHOOL OF ENGINEERING. Inventor of the week archive. Disponível em: <<http://web.mit.edu/invent/iow/metcalfe.html>> Acesso em: 26 jan. 2004.
- [20] SPURGEON's, C. 100 Mbps Fast Ethernet. Disponível em: <<http://www.ethermanage.com/ethernet/100mbps.html>>. Acesso em: 19 jan. 2004.
- [21] MOURA, A. S., Gigabit Ethernet: novo horizonte. Rede Nacional de Ensino e Pesquisa, v. 2, Disponível em: <<http://www.rnp.br/newgen/9802/gbe-intr.html>>. Acesso em: 15 jan. 2004.
- [22] OTAKI, D. T. BICSI2001. Disponível em: <<http://www.bicsi.com.br/download/bicsi2001/br0105.pdf>> Acesso em: 14 jan. 2004.
- [23] POTTER, D. Using Ethernet for industrial I/O data acquisition. In: INSTRUMENTS AND MEASUREMENT TECHNOLOGY CONFERENCE, 1999. **Conference...** 1999 v. 3, p. 1492 – 1496.
- [24] LEE, S-H.; CHO, K-H. Congestion control of high-speed Gigabit-Ethernet networks for industrial application. In: INTERNATIONAL SYMPOSIUM INDUSTRIAL ELECTRONIC, ISIE, 2001, Pusan. **Proceedings of the...** Pusan: IEEE, 2001. v. 1, p. 260-265.
- [25] DEPARTAMENTO DE ELECTRÓNICA INDUSTRIAL – Universidade do Minho, Portugal. A Ethernet na automação industrial. Disponível em: <www.dei.uminho.pt/mei/Sistreal/RT11.pdf>. Acesso em: 03 jan. 2004.
- [26] PlanetSab. Java. Disponível em: <http://www.planetsap.com/thesap_java.htm> Acesso em: 12 jan. 2004.
- [27] Fundão Inc. Introdução ao mundo Java. Disponível em: <<http://www2.fundao.pro.br/articles.asp?cod=58>> Acesso em: 11 jan. 2004
- [28] LINDHOLM, T.; YELLIN, F. The Java™ virtual machine specification, Sun Microsystems. Disponível em: <<http://java.sun.com/docs/books/vmspec/html/VMSpecTOC.doc.html>>. Acesso em: 11 jan. 2004.
- [29] DEITEL, H. M.; DEITEL, P. J. Java como programar. 3 ed. Porto Alegre: Bookman, 2001. 1164 p.



- [30] WOLLRATH, A.; WALDO, J. The Java tutorial. RMI. Disponível em: <<http://java.sun.com/docs/books/tutorial/rmi/>> Acesso em: 18 dez. 2003.
- [31] SUN MICROSYSTEMS. Core Java – Java communications. Disponível em: <<http://java.sun.com/products/javacomm/index.jsp>>. Acesso em: 27 jan. 2004.
- [32] DotLinux Solutions, Comunidade Java do Brasil é a maior do mundo. Disponível em: <<http://www.dotlinux.net/modules.php?name=News&file=article&sid=418>>. Acesso em: 06 abril 2004.
- [33] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. IEEE 1451 website. What is IEEE 1451? introduction. Disponível em: <<http://ieee1451.nist.gov>>. Acesso em: 01 dez 2003.
- [34] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. IEEE 1451 website. What is IEEE 1451? objective. Disponível em: <<http://ieee1451.nist.gov>> . Acesso em: 01 dez 2003.
- [35] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. IEEE 1451 website. What is IEEE 1451? history. Disponível em: <http://ieee1451.nist.gov>>. Acesso em: 01 dez 2003.
- [36] LEE, K. IEEE 1451: A standard in support of smart transducer networking. In: INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE-IMTC, SMART CONNECTIVITY INTEGRATING MEASUREMENT AND CONTROL, 17, 2000, Baltimore. **Proceedings of the...**, Baltimore: IEEE, 2000, v. 2, p. 525 – 528.
- [37] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. IEEE 1451 website. What is IEEE 1451? evolution. Disponível em: <<http://ieee1451.nist.gov>>. Acesso em: 01 dez 2003.
- [38] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. IEEE 1451 Working group - 1451.2. Disponível em: <<http://ieee1451.nist.gov>>. Acesso em: 01 dez 2003.
- [39] LEE, K.B.; Schneeman, R.D. Internet-based distributed measurement and control. New York, **Applications Instrumentation & Measurement Magazine**, IEEE , v. 2, p.23-27, jun. 1999.



- [40] ZELENOVSKY, R.; MENDONÇA, A., **Hardware e interfaceamento**. Rio de Janeiro: MZ, 1999. 753 p.
- [41] LOZANO, F. O que é software livre. Disponível em:
<<http://www.gnu.org/philosophy/free-sw.pt.html>>. Acesso em: 25 jan. 2004.



APÊNDICE A - Sites Para Assunto Java

Neste Apêndice, mostram-se alguns sites dedicados ao assunto Java.

<http://java.sun.com> O site de Java pertencente a *Sun Microsystems, Inc.* é uma opção essencial para quem deseja utilizar a tecnologia Java. Este site também oferece um recurso completo com notícias, informações, suporte, cursos, etc.

<http://www.gamelan.com> Este site apresenta uma lista de discussão sobre Java. No site pode-se descarregar programas gratuitamente, fazer perguntas a especialistas, grupos de discussão e possui um glossário de terminologia. Além disso, informa sobre eventos, diretórios especializados de indústria e centenas de recursos relacionados com Java.

www.javamagazine.com.br/ Este site é dedicado a primeira revista sobre Java no Brasil. Neste site encontra-se várias dicas de Java, possibilidade de assinatura da revista, etc.

<http://www.mundojava.com.br/> Site pertencente a revista MundoJava, traz artigos sobre Java e possibilidade de assinatura da revista.

<http://www.javaworld.com> A revista JavaWorld on-line, que possui recursos para informações sobre Java.

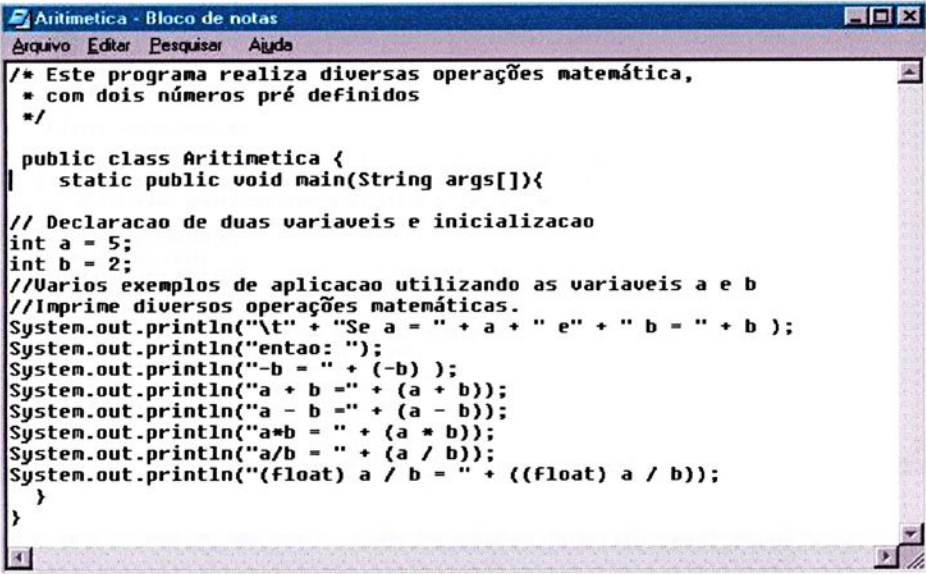
<http://www.javaman.com.br/> Este site é um dos principais na divulgação da tecnologia Java no Brasil.

<http://www.soujava.org.br/index.jsp> Este site apresenta o que acontece sobre Java no Mundo. Possui um grupo de discussão, projetos com empresas, vagas de emprego e pode-se descarregar recursos Java.



APÊNDICE B – Programas em Java.

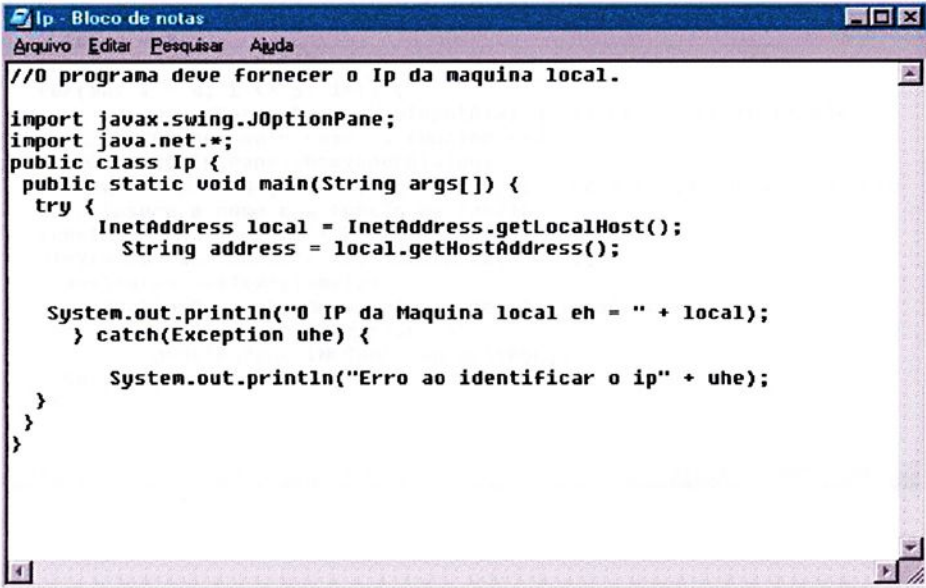
Neste Apêndice apresentam-se alguns exemplos de programação utilizando a linguagem Java



```
/* Este programa realiza diversas operações matemática,
 * com dois números pré definidos
 */

public class Aritimetica {
    static public void main(String args[]){

// Declaracao de duas variaveis e inicializacao
int a = 5;
int b = 2;
//Varios exemplos de aplicacao utilizando as variaveis a e b
//Imprime diversos operações matemáticas.
System.out.println("\t" + "Se a = " + a + " e" + " b = " + b );
System.out.println("\t\tentao: ");
System.out.println("\t-b = " + (-b) );
System.out.println("\ta + b = " + (a + b));
System.out.println("\ta - b = " + (a - b));
System.out.println("\ta*b = " + (a * b));
System.out.println("\ta/b = " + (a / b));
System.out.println("\t(float) a / b = " + ((Float) a / b));
    }
}
```



```
//O programa deve fornecer o Ip da maquina local.

import javax.swing.JOptionPane;
import java.net.*;

public class Ip {
    public static void main(String args[]) {
        try {
            InetAddress local = InetAddress.getLocalHost();
            String address = local.getHostAddress();

            System.out.println("\tO IP da Maquina local eh = " + local);
        } catch (Exception uhe) {
            System.out.println("\tErro ao identificar o ip" + uhe);
        }
    }
}
```

```
Adiciona3Numeros - Bloco de notas
Arquivo Editar Pesquisar Ajuda
// Programa faz a adiç3o de tr3s n3meros inteiros

import javax.swing.JOptionPane;

public class Adiciona3Numeros {
    public static void main(String args[]) {

        String A;
        String B;
        String C;
        String output = " ";
        int a;
        int b;
        int c;
        int R;

        // Entre com os valores

        A = JOptionPane.showInputDialog ("Digite o primeiro n3mero:");
        B = JOptionPane.showInputDialog ("Digite o segundo n3mero:");
        C = JOptionPane.showInputDialog ("Digite o terceiro n3mero:");

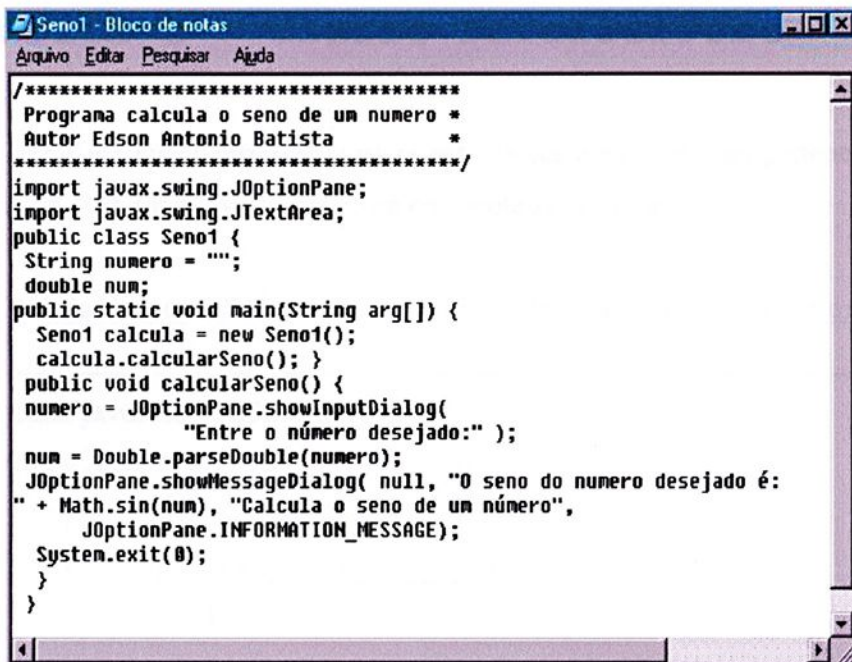
        a = Integer.parseInt(A);
        b = Integer.parseInt(B);
        c = Integer.parseInt(C);

        R = a + b + c;
        output = "O resultado da soma dos tr3s valores 3: " + R;
        JOptionPane.showMessageDialog(null, output,
            "Soma de tr3s n3meros",
            JOptionPane.INFORMATION_MESSAGE);
        System.exit(0);
    }
}
```

```
ListaFuncionarios2 - Bloco de notas
Arquivo Editar Pesquisar Ajuda

import javax.swing.JOptionPane;
import javax.swing.JTextArea;

public class ListaFuncionarios2 {
    public static void main (String args[]) {
        String nome; String funcao; String tabela = " ";
        int c = 0;
        tabela += "Nome\tFunç3o";
        for(int i = 0; i <= 5; i++) {
            nome = JOptionPane.showInputDialog( // L3 o nome funcion3rio
                "Entre com o nome do funcion3rio");
            funcao = JOptionPane.showInputDialog(
                "Entre com a funç3o do funcion3rio");// L3 a funç3o do funcion3rio
            // Insere o nome e a funç3o na tabela
            tabela += "\n" + nome + "\t" + funcao; c = i; }
        JTextArea tabelaArea = new JTextArea(6, 15);
        tabelaArea.setText(tabela);
        JOptionPane.showMessageDialog(null, tabelaArea,
            "Tabela de funcion3rios",
            JOptionPane.INFORMATION_MESSAGE);
        System.exit(0);
    }
}
```

```
Sen01 - Bloco de notas
Arquivo  Editar  Pesquisar  Ajuda

/*****
Programa calcula o seno de um numero *
Autor Edson Antonio Batista          *
*****/
import javax.swing.JOptionPane;
import javax.swing.JTextArea;
public class Sen01 {
    String numero = "";
    double num;
    public static void main(String arg[]) {
        Sen01 calcula = new Sen01();
        calcula.calcularSeno(); }
    public void calcularSeno() {
        numero = JOptionPane.showInputDialog(
            "Entre o número desejado:" );
        num = Double.parseDouble(numero);
        JOptionPane.showMessageDialog( null, "O seno do numero desejado é:
" + Math.sin(num), "Calcula o seno de um número",
            JOptionPane.INFORMATION_MESSAGE);
        System.exit(0);
    }
}
```

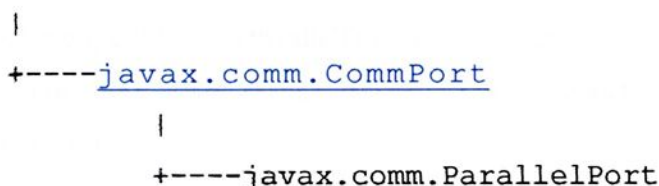
APÊNDICE C - Classe ParallelPort

Neste Apêndice apresentam-se as sub-classes e as interfaces pertencentes à super-classe ParallelPort. Esta super-classe, faz parte do pacote Java Comm.

[All Packages](#) [Class Hierarchy](#) [This Package](#) [Previous](#) [Next](#) [Index](#)

Class javax.comm.ParallelPort

java.lang.Object



public abstract class **ParallelPort**

extends [CommPort](#)

A parallel communications port. ParallelPort describes the low-level interface to a parallel communications port made available by the underlying system. ParallelPort defines the minimum required functionality for parallel communications ports.

See Also:

[CommPort](#), [CommPortIdentifier](#)

Variable Index

•[LPT MODE ANY](#)

Picks the best available mode.

•[LPT MODE ECP](#)

Enhanced capabilities port.

•[LPT MODE EPP](#)

Extended parallel port.



•LPT MODE NIBBLE

Nibble Mode.

•LPT MODE PS2

Byte mode.

•LPT MODE SPP

Compatibility mode.

Constructor Index

•ParallelPort()

Method Index

•addEventListener(ParallelPortEventListener)

Registers a ParallelPortEventListener object for parallelPortEvents.

•getMode()

Gets the currently configured mode.

•getOutputBufferFree()

Gets the number of bytes available in the output buffer.

•isPaperOut()

Checks if the port is indicating an "Out of Paper" state.

•isPrinterBusy()

Checks if the port is indicating a "Printer Busy" state.

•isPrinterError()

Checks if the printer has encountered an error.

•isPrinterSelected()

Checks if the printer is in selected state.

•isPrinterTimedOut()

Checks if the printer has timed out.

•notifyOnBuffer(boolean)

Expresses interest in being notified when the output buffer is empty.

•notifyOnError(boolean)

Expresses interest in being notified of port errors.

•removeEventListener()



Deregisters event listener registered using `addEventListener`.

• **restart()**

Restarts output after an error.

• **setMode(int)**

Sets the printer port mode.

• **suspend()**

Suspends output.

Variables

● **LPT_MODE_ANY**

```
public static final int LPT_MODE_ANY
```

Picks the best available mode.

● **LPT_MODE_SPP**

```
public static final int LPT_MODE_SPP
```

Compatibility mode. Unidirectional.

● **LPT_MODE_PS2**

```
public static final int LPT_MODE_PS2
```

Byte mode. Bi-directional.

● **LPT_MODE_EPP**

```
public static final int LPT_MODE_EPP
```

Extended parallel port.

● **LPT_MODE_ECP**

```
public static final int LPT_MODE_ECP
```

Enhanced capabilities port.

● **LPT_MODE_NIBBLE**

```
public static final int LPT_MODE_NIBBLE
```

Nibble Mode. Bi-directional. HP Bi-tronics. 4 bits at a time.

Constructors

● **ParallelPort**

```
public ParallelPort()
```

Methods



●addEventListener

public abstract void addEventListener([ParallelPortEventListener](#) lsnr) throws TooManyListenersException.

Registers a ParallelPortEventListener object for ParallelPortEvents. Interest in specific events may be expressed by using the notifyOnError and notifyOnBuffer methods.

Only one listener per ParallelPort is supported. Calling addEventListener multiple times will simply replace the current ParallelPortEventListener object.

After the port is closed, no more event will be generated. Another call to open() of the port's CommPortIdentifier object will return a new CommPort object, and the lsnr has to be added again to the new CommPort object to receive event from this port.

Parameters: lsnr - A ParallelPortEventListener object whose parallelEvent method will be called to indicate an event.

Throws: TooManyListenersException

If an initial attempt to attach a listener succeeds, subsequent attempts will throw TooManyListenersException without effecting the first listener.

●removeEventListener

public abstract void removeEventListener()

Deregisters event listener registered using addEventListener. This is done automatically when a port is closed.

●notifyOnError

public abstract void notifyOnError(boolean notify)

Expresses interest in being notified of port errors.

Parameters: notify - enable or disable error notification based on whether notify is true or false.

●notifyOnBuffer

public abstract void notifyOnBuffer(boolean notify)

Expresses interest in being notified when the output buffer is empty.

Parameters: boolean - true enables notification; false disables notification.

●getOutputBufferFree

public abstract int getOutputBufferFree()



Gets the number of bytes available in the output buffer.

Returns: integer value corresponding to the number of bytes available in the output buffer.

● **isPaperOut**

```
public abstract boolean isPaperOut()
```

Checks if the port is indicating an "Out of Paper" state.

Returns: true if printer has run out of paper, false if printer does not report "Out of paper".

● **isPrinterBusy**

```
public abstract boolean isPrinterBusy()
```

Checks if the port is indicating a "Printer Busy" state.

Returns: true if printer is busy, false if printer is idle

● **isPrinterSelected**

```
public abstract boolean isPrinterSelected()
```

Checks if the printer is in selected state.

Returns: true if printer is selected, false if printer is in deselected mode

● **isPrinterTimedOut**

```
public abstract boolean isPrinterTimedOut()
```

Checks if the printer has timed out.

Returns: true if printer has timed out, false if printer does not report "Out of paper".

● **isPrinterError**

```
public abstract boolean isPrinterError()
```

Checks if the printer has encountered an error.

Returns: true if printer has encountered an error, false if printer does not report "Out of paper".

● **restart**

```
public abstract void restart()
```

Restarts output after an error.

● **suspend**

```
public abstract void suspend()
```



Suspends output.

● **getMode**

```
public abstract int getMode()
```

Gets the currently configured mode.

Returns: mode returned is one of LPT_MODE_SPP, LPT_MODE_PS2, LPT_MODE_EPP, or LPT_MODE_ECP

● **setMode**

```
public abstract int setMode(int mode)
```

throws [UnsupportedCommOperationException](#)

Sets the printer port mode.

Parameters: mode - one of LPT_MODE_ANY, LPT_MODE_SPP, LPT_MODE_PS2, LPT_MODE_EPP, or LPT_MODE_ECP

Throws: [UnsupportedCommOperationException](#)

Is thrown if the mode is not supported by the underline driver.



APÊNDICE D - Classe IEEE1451.1

Neste Apêndice mostra-se a hierarquia de classe em conformidade com o padrão IEEE1451.1.

Root

Entity

Block

NCAP Block

Function Block

Base Transducer Block

Transducer Block

Dot2 Transducer Block

Dot3 Transducer Block

Dot4 Transducer Block

Component

Parameter

Parameter Whit Update

Physical Parameter

Scalar Parameter

Scalar Series Parameter

Vector Parameter

Vector Series Parameter

Time Parameter

Action

File

Partitioned File

Component Group

Service

Base Port

Base Client Port

Client Port

Asynchronous Client Port

Base Publisher Port

Publisher Port

Self Identifying Publisher Port

Event Generator Publisher Port

Subscriber port

Mutex Service

Condition Variable Service





UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Câmpus de Ilha Solteira
Programa de Pós-Graduação em Engenharia Elétrica
Av. Brasil Centro, 56
15385-000 Ilha Solteira - SP
www.dee.feis.unesp.br

