UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
(Bacharelado)

PROTÓTIPO DE UM HARDWARE PARA CONTROLE DE FREQUÊNCIA ACADÊMICA

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERNANDO LUIZ MELATI DA SILVA

BLUMENAU, MARÇO/2002

PROTÓTIPO DE UM HARDWARE PARA CONTROLE DE FREQUÊNCIA ACADÊMICA

FERNANDO LUIZ MELATI DA SILVA

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Miguel Alexandre Wisintainer — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Miguel Alexandre Wisintainer

_____

Prof. Antônio Carlos Tavares

_____

Prof. Francisco Adell Péricas

DEDICATÓRIA

Aos meus pais José Itamar e Maria Teresa, minha namorada Shirley pelo carinho, compreensão e incentivo sem os quais este trabalho não seria realidade.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

RESUMO

O presente trabalho apresenta considerações sobre a automação do processo de controle de freqüência acadêmico, conceitos sobre arquitetura TCP/IP, principalmente os protocolos TCP e SMTP que foram utilizados na construção do sistema de controle de freqüência. Apresenta também detalhes da especificação e desenvolvimento (hardware / software) do sistema.

ABSTRACT

This work presents considerations about academic frequency automation process, concepts about TCP/IP architecture, specificly the SMTP protocol that was used in the building of the frequency control system. It also presents details about specification and system´s development (hardware / software).

1 INTRODUÇÃO

As universidades são centros de criação, transmissão e difusão da cultura, da ciência e da tecnologia, que, através da articulação do estudo, da docência e da investigação, se integram na vida da sociedade (Marcovitch, 1998).

São fins das universidades:

a) a formação humana, cultural, científica e técnica;
b) a realização de investigação fundamental e aplicada;
c) a prestação de serviços à comunidade, numa perspectiva de valorização recíproca;
d) o intercâmbio cultural, científico e técnico com instituições congêneres nacionais e estrangeiras;
e) a contribuição, no seu âmbito de atividade, para a cooperação internacional e para a aproximação entre os povos.

Às universidades compete a concessão de graus e títulos acadêmicos e honoríficos, de outros certificados e diplomas, bem como a concessão de equivalência e o reconhecimento de graus e habilitações acadêmicos (Marcovitch, 1998).

Nas universidades são formados os futuros profissionais que atuarão na sociedade. No entanto, a universidade deve monitorar o aprendizado dos mesmos. Uma forma de monitoração de aprendizagem são as verificações. Através delas, as universidades garantem a formação de profissionais qualificados (Marcovitch, 1998).

A verificação de aprendizagem, abrangendo os aspectos de assiduidade e aproveitamento, ambos eliminatórios por si mesmos, é realizada:

a) por disciplina, nos cursos de graduação e pós-graduação;
b) global, de toda a matéria, nos cursos de extensão.

Entende-se por assiduidade a freqüência às atividades de cada disciplina, considerando-se nela reprovado o aluno que deixar de comparecer a, no mínimo uma determinada fração da sua carga horária. Como exemplo, no caso da FURB-Universidade Regional de Blumenau, este mínimo é de setenta e cinco por cento (75%) da carga horária da mesma, vedado o abono de faltas (Furb, 1995).

Seguindo os rumos da tecnologia, as universidades buscam aprimoramento tecnológico, tanto em nível acadêmico como administrativo. Tal aprimoramento pode ser observado na automação de certos processos, como vídeo conferências entre universidades, cartões de acesso às bibliotecas, *intranet* permitindo acesso à internet e compartilhamento de recursos entre departamentos.

1.1 OBJETIVOS DO TRABALHO

Neste trabalho tem-se como objetivo desenvolver um *software* para microcomputador e outro para um *hardware* baseado em microcontrolador, para controle de freqüência de alunos. O aluno utilizará o cartão da biblioteca para, através de um leitor de cartão (código de barras), fornecer o código ao microcontrolador, indicando sua presença em sala de aula. Este microcontrolador estará conectado a *internet* por meio de um cabo par trançado. E finalmente, por meio exclusivo do protocolo SMTP (*Simple Mail Transfer Protocol*) rodando sobre o TCP/IP, enviará um e-mail ao professor com a lista de presenças daquela aula, ao término da mesma.

Os objetivos específicos do trabalho são:

a) armazenamento do código do aluno/professor e horário de acesso no microcontrolador;
b) acesso à base de dados da universidade, para recuperar nomes dos códigos armazenados e *e-mail* do professor;
c) montagem e envio do e-mail pelo microcontrolador;

1.2 ESTRUTURA DO TRABALHO

O primeiro capítulo (presente) é destinado à introdução, objetivos e estrutura do trabalho.

O segundo capítulo é dedicado ao estudo da arquitetura TCP/IP. São apresentados conceitos, características, principais protocolos e um estudo mais detalhado sobre o protocolo SMTP.

O terceiro capítulo descreve o kit de desenvolvimento Rabbit 2000 TCP/IP, composto de placa programável e acessórios.

O quarto capítulo enfoca a especificação do sistema.

O capítulo cinco enfoca a implementação do sistema.

No capítulo seis são relatadas as conclusões obtidas no trabalho, dificuldades encontradas e propostas para futuras implementações.

2 TCP/IP – TRANSMISSION CONTROL PROTOCOL / INTERNET PROTOCOL

A história do conjunto de protocolos TCP/IP, segundo Dumas(1995), pode ser traçada ao se referir a uma das primeiras redes remotas, que consistia de computadores de diferentes fabricantes, os quais rodavam diferentes sistemas operacionais.

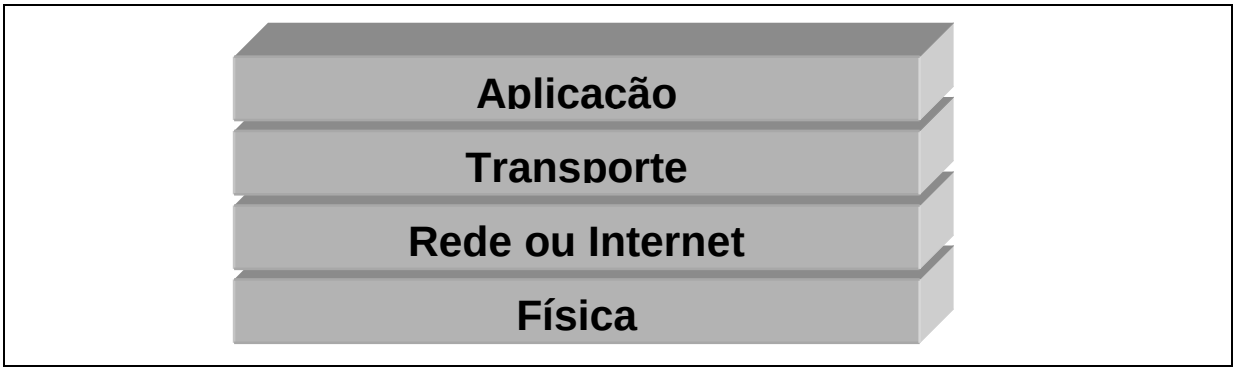
De fato, segundo Gasparini(1994), o princípio básico da arquitetura TCP/IP é oferecer aos usuários e suas aplicações, um conjunto de ferramentas que possibilite a comunicação entre redes e equipamentos diversos, escondendo os detalhes físicos (hardware e protocolos de nível inferior) dos mesmos. Desta forma, quando se implementa TCP/IP em uma ou mais redes interconectadas, está-se apenas adicionando a estas um conjunto de protocolos (conjunto de regras e convenções que rege a troca de informações entre computadores) e aplicações padronizadas, mantendo as características anteriores do hardware já existente.

Os protocolos da arquitetura TCP/IP podem ser utilizados sobre qualquer estrutura de rede, seja ela simples como uma ligação ponto-a-ponto ou uma rede de pacotes complexa. Como exemplo, pode-se empregar estruturas de rede como *Ethernet*, *Token-Ring*, FDDI, PPP, ATM, X.25, *Frame-Relay*, barramentos SCSI, enlaces de satélite, ligações telefônicas discadas e várias outras como meio de comunicação do protocolo TCP/IP (Werner, 1999).

2.1 CAMADAS DA ARQUITETURA TCP/IP

A arquitetura TCP/IP estabelece uma estrutura em quatro camadas ou níveis conforme é mostrado na figura 2.1 (Chiozzotto, 1999).

FIGURA 2.1 – CAMADAS DA ARQUITETURA TCP/IP



2.1.1 CAMADA TRANSPORTE

Esta camada tem como principal objetivo prover a transferência confiável de dados entre processos de diferentes estações pertencentes ou não à mesma rede, garantindo que os dados sejam entregues livres de erro e em seqüência, sem perdas ou duplicação (Carvalho, 1994).

Como esta camada serve de transporte a várias aplicações simultaneamente, ela deve controlar vários canais lógicos distintamente (Gasparini, 1994).

Na Arquitetura *Internet* , segundo Carvalho(1994), são definidos dois tipos de protocolo de transporte: o protocolo orientado à conexão (TCP) e o protocolo não orientado à conexão (UDP), sendo o primeiro muito mais sofisticado e eficiente.

Transmission Control Protocol (TCP) – foi idealizado para proporcionar uma forma segura de transferência de dados provenientes de diversas aplicações, utilizando vários meios de comunicação (nível físico). O TCP foi criado para fornecer uma entrega de dados de forma virtual, garantindo a integridade dos mesmos, portanto adotando um sistema de controle de fluxo e checagem de erros (Gasparini, 1994).

User Datagram Protocol (UDP) – segundo Gasparini (1994), este protocolo foi desenvolvido para utilização de aplicações que não devem gerar um alto volume de tráfego na Internet. Em comparação ao TCP, o UDP é muito inferior, porém com

overhead muito menor. O UDP não implementa confirmações e seqüenciamento, portanto é considerado não confiável, sem conexão, etc.

Para permitir identificar uma aplicação em um dado sistema, é empregado o conceito de *ports* (portas) na camada de transporte, que é representada por um número inteiro associado à aplicação a partir da negociação com o sistema operacional. Aplicações padronizadas possuem números de portas, atribuídos pelo IAB (Internet Activities Board), que são universalmente conhecidos dentro de uma rede Internet, não importando em que sistema estejam sendo processadas. Como exemplo pode-se citar: o correio eletrônico implementado pelo SMTP (Simple Mail Transfer Protocol), que utliza a porta 25; e o FTP (File Transfer Protocol) que utiliza duas portas, sendo a 20 dedicada a conexão de dados e a porta 21 à conexão de controle.

Em seguida, são apresentados os conceitos e terminologias relacionados à *socket*, que é o mecanismo utilizado para comunicação das aplicações com a camada de transporte (TCP e UDP).

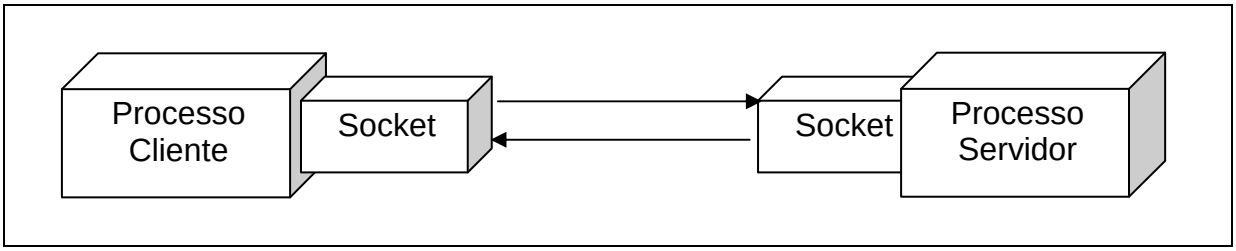
2.1.1.1 SOCKET

Segundo Carvalho(1994), para uma aplicação (processo) de uma estação (*host*) utilizar os serviços do TCP/UDP, ela se associa a uma porta (*port*) desse serviço. Uma porta, quando ligada a um endereço IP, constitui um *socket*. Dentro de uma estação cada porta é identificada por um número próprio e único; de maneira análoga, cada estação tem um endereço IP único dentro de uma rede *internet*. Daí decorre que um *socket* tem uma identificação única dentro de uma rede *internet*.

Um *socket* pode ser considerado um ponto de referência para o qual as mensagens podem ser enviadas e a partir do qual as mensagens são recebidas. Um par de *sockets* é usado para identificar uma conexão entre dois processos. Umas vez estabelecida uma conexão, cada *socket* passa a corresponder a um ponto final (*endpoint*) dessa conexão (Carvalho, 1994).

A figura 2.2 mostra um exemplo de comunicação via *Socket* TCP/IP.

FIGURA 2.2 – EXEMPLO DE COMUNICAÇÃO VIA *SOCKET* TCP/IP



2.1.2 CAMADA APLICAÇÃO

Nessa camada residem as aplicações que fazem uso dos protocolos da camada TRANSPORTE para comunicação por meio de uma internet, ou seja, essas aplicações usufruem do resultado final de toda a transparência fornecida pela arquitetura TCP/IP (Chiozzotto, 1999).

Segundo Gasparini(1994) a camada APLICAÇÃO tem por função efetuar acondicionar programas que utilizam a rede. É responsável pela interface com as aplicações dos *host*. Nessa camada encontramos aplicações como correio eletrônico, transferência de arquivos, emuladores de terminais, etc.

2.1.2.1 SMTP – SIMPLE MAIL TRANSFER PROTOCOL

Este protocolo viabiliza uma das facilidades de comunicação de dados implementada pela Arquitetura Internet TCP/IP, o Correio Eletrônico.

Diversos ambientes computacionais implementam facilidades de Correio Eletrônico através de redes de teleprocessamento. Nesses ambientes é comum o acesso de usuários apenas para uso dessa facilidade (*Eletronic Mail*). As aplicações de Correio Eletrônico são largamente difundidas e utilizadas, pois oferecerem um serviço simples e rápido de transferência de mensagens, minimizando o uso de telefone e papel dentro de uma corporação. Esta facilidade pode ser aplicada para pequenas mensagens ou até extensos memorandos. Na verdade é mais comum encontrar

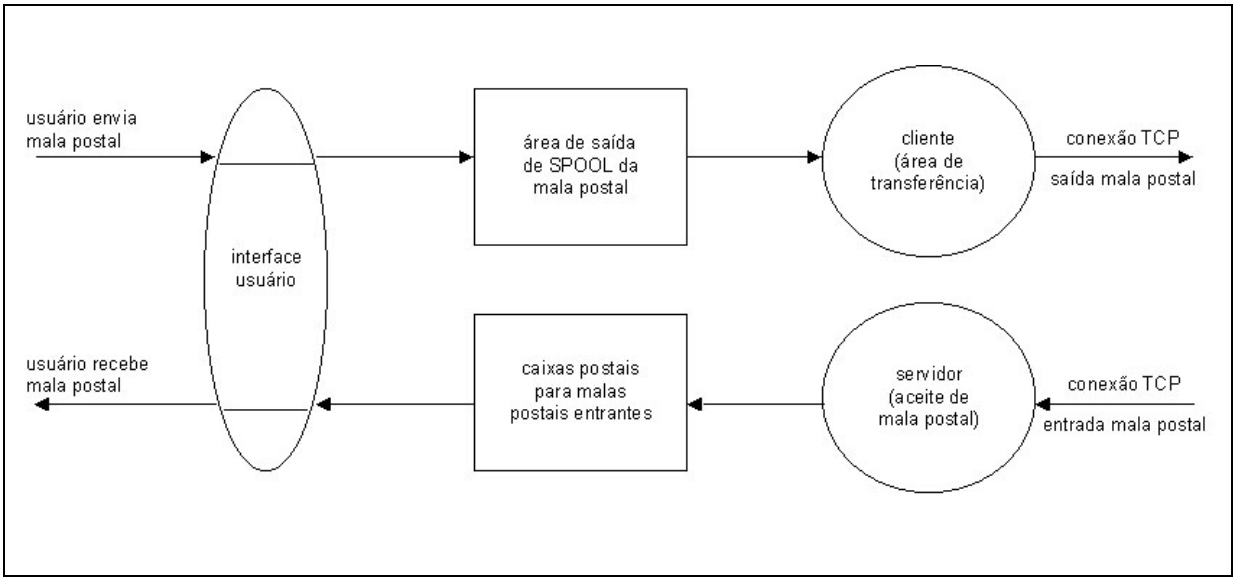
usuários que transmitem arquivos via Correio Eletrônico do que os que utilizam softwares de Transferência de Arquivos. Diferente dos demais programas de comunicação de dados (FTP, TFTP, TELNET, etc.), na utilização de um Correio Eletrônico, o transmissor não deseja aguardar que determinada máquina destinatária esteja disponível, como também não deseja que a operação de transmissão de mensagens seja interrompida por falhas na rede ou na máquina receptora (Gasparini, 1994).

Enquanto os demais protocolos da camada de aplicação da Arquitetura Internet efetuam comunicações de forma "*on line*", verificando se o destino está ativo ou inativo, implementando métodos de detecção e correção de erros, "*time out*", retransmissões, etc., o SMTP envia sua mensagem a uma máquina "*server*" que deve armazenar a mesma em uma área denominada "*spool*", e após isto, efetuar a entrega da mensagem ao respectivo destinatário. Esta técnica conhecida como "*Spooling*", baseia-se também na filosofia "client-server", onde o transmissor é um cliente, que envia sua mensagem a um servidor. O servidor recebe a mensagem e coloca a mesma em uma área de armazenamento (*spool*), junto com a identificação do transmissor (remetente) da máquina destinatária e o tempo de armazenamento (Gasparini, 1994).

O servidor então inicia uma conexão com a máquina destino, efetuando a transferência da mensagem, sendo que após a conclusão desta etapa, efetua a confirmação de entrega da mesma ao cliente, "limpando ou não o *spool*". Desta forma, o cliente envia sua mensagem rapidamente, ficando livre para executar outras tarefas.

Este processo de troca de mensagens é ilustrado na figura 2.3.

FIGURA 2.3 – PROCESSO DE TROCA DE MENSAGENS



Quando a conexão TCP não é possível (por indisponibilidade da rede, do servidor remoto, etc.), o processo de transferência registra o horário da tentativa a aguarda um "*timer*" para nova conexão. Na verdade, existe um processo periódico de verificação da área de *spool* para transmissões pendentes. O processo de transferência tente transmitir estas mensagens continuamente em intervalos de tempo predeterminados até expirar um número de tentativas ou um período de tempo configurado (este método é o mais utilizado), como uma hora, um dia, ou uma semana (Gasparini, 1994).

2.1.2.1.1 – FORMATO DO ENDEREÇO

Segundo Chiozzotto(1999), a determinação do destino de uma dada mensagem é realizada a partir de um endereço associado a cada usuário, conhecido como *endereço de correio eletrônico* (*e-mail address*). O formato desse endereço é especificado na RFC 822, e por isso é também conhecido como *RFC 822 e-mail address.*

Para usuários de redes de arquitetura TCP/IP, esses endereços assumem um formato simples, do tipo **user@host**, onde *user* especifica um nome para a caixa

postal de um usuário (normalmente o nome do usuário) e *host* especifica o equipamento em que reside essa caixa postal.

Com a utilização de DNS para resolução de nomes de domínio, é possível especificar, no arquivo de configuração de cada zona, um registro que informa os nomes de equipamentos responsáveis pelo tratamento de mensagens de correio dos domínios dessa zona.

Feito isso, o endereço de correio pode assumir o formato que é usado na Internet, do tipo **user@domain**, de forma que os usuários que possuam caixa postal em qualquer equipamento de um domínio têm os seus endereços representados de forma independente do equipamento no qual suas caixas postais realmente residem.

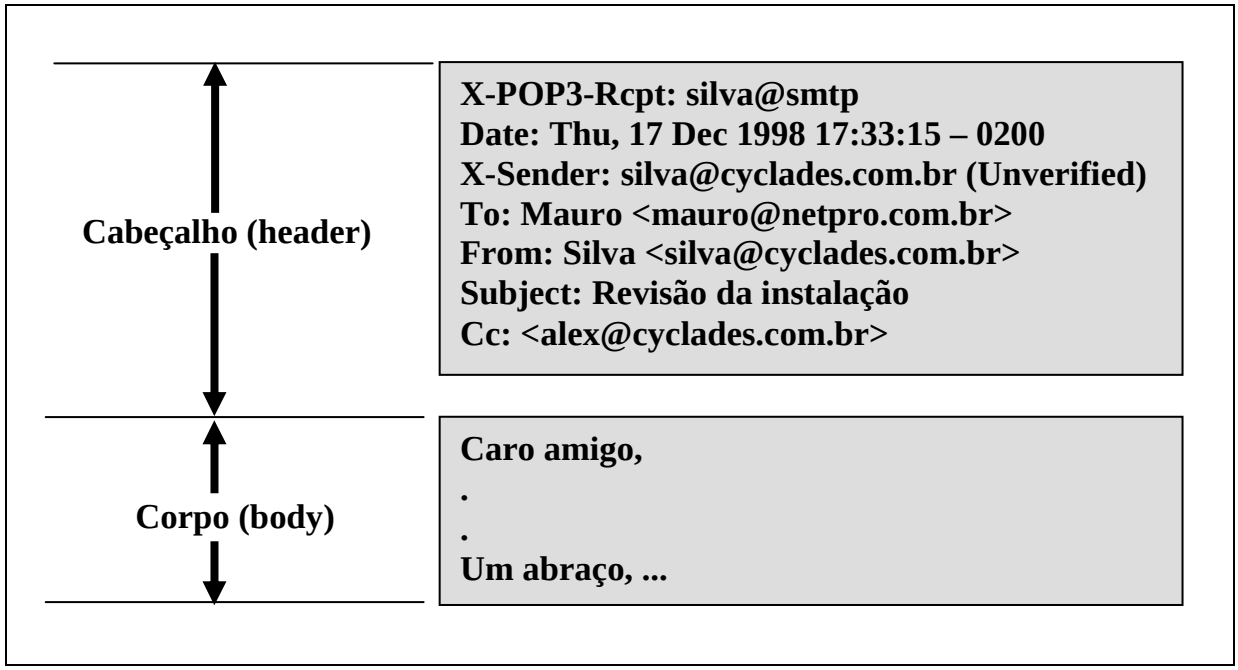
2.1.2.1.2 – FORMATO DAS MENSAGENS

Segundo Chiozzotto(1999), as mensagens de correio eletrônico possuem, além do texto fornecido pelo usuário, informações adicionais necessárias para funcionamento do processo de transmissão.

A mensagem, portanto, é composta de duas partes: um *cabeçalho*, em que constam essas informações adicionais, e um *corpo* com o texto da mensagem fornecido pelo usuário emitente da mensagem, separados por uma linha em branco.

De acordo com a RFC 822 (1982), uma mensagem tem o formato como descrito na figura 2.4.

FIGURA 2.4 – FORMATO DAS MENSAGENS



2.1.2.1.3 – FUNCIONAMENTO

O protocolo SMTP, como o nome já diz, é relativamente simples; sua implementação prevê o seu uso em um modelo Cliente / Servidor da aplicação, usando o protocolo TCP da camada Transporte para a comunicação entre os programas cliente e servidor. O programa servidor espera conexões TCP no *end-point* de número de *port* igual a 25, e o cliente usa um *end-point* com um número de *port* qualquer.

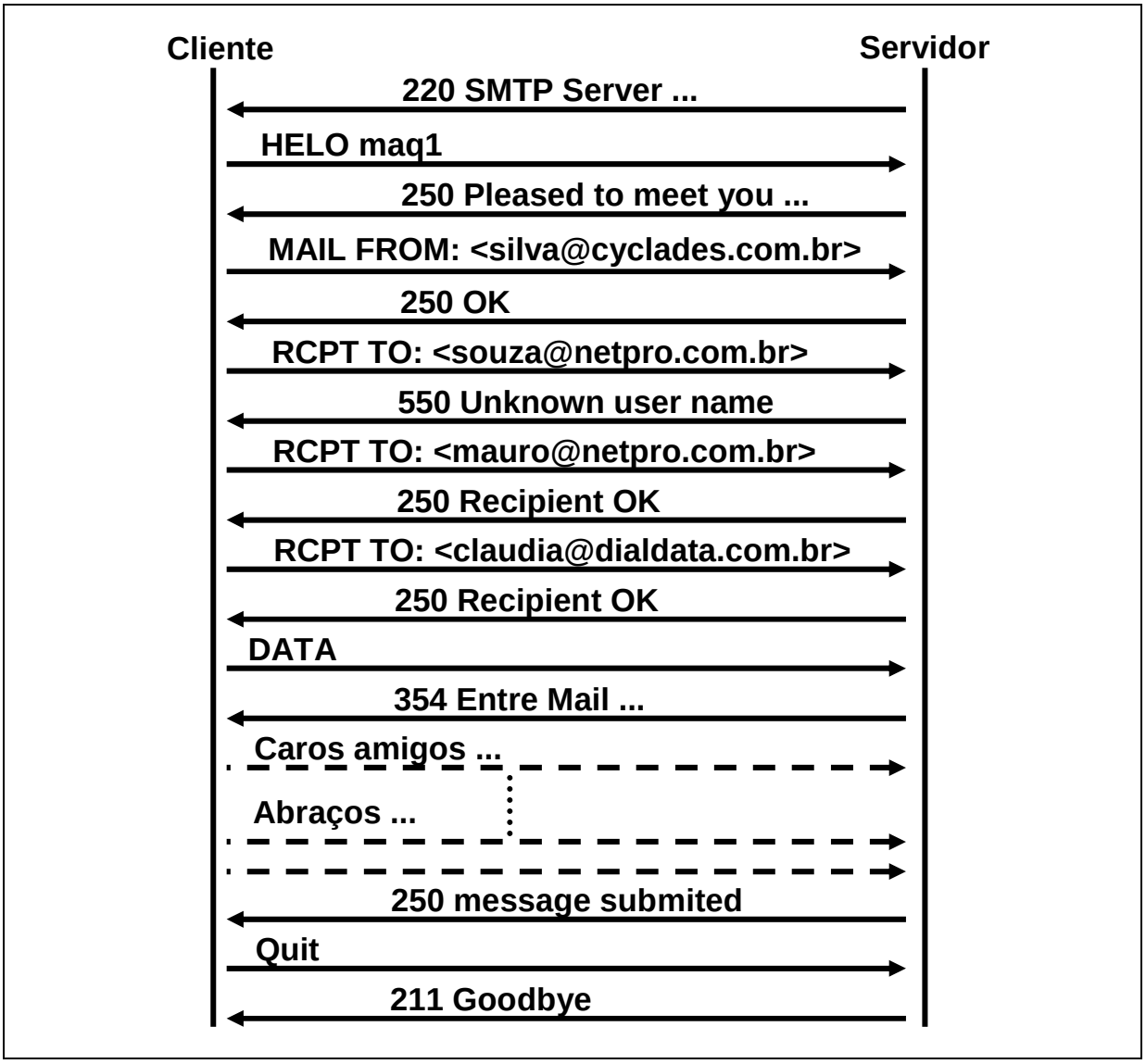
A tabela 2.1 contém os principais comandos implementados no protocolo SMTP para troca de mensagens e suas descrições. (Carvalho, 1994)

TABELA 2.1 – DESCRIÇÃO DOS PRINCIPAIS COMANDOS PARA TROCA DE MENSAGENS UTILIZANDO O PROTOCOLO SMTP

Comando	Significado
HELO	**Identifica o emissor da mensagem para o receptor.**
MAIL FROM	**Inicializa uma transação de *mail* na qual uma mensagem é enviada a uma ou mais caixas de mensagens (*mailbox*), informando a origem da mensagem.**
RCPT TO	**Identifica o destinatário (*mailbox*) da mensagem; múltiplos destinatários são definidos por múltiplos usos desse comando.**
DATA	**Inicializa a transmissão da mensagem, após seu uso é transmitido o conteúdo da mensagem, que pode conter qualquer um dos 128 caracteres ASCII, O seu término é especificado por uma seqüência especial de caracteres (<CRLF>.<CRLF>, isto é, uma linha contendo apenas um ponto).**
TURN	**Inverte o sentido da conexão, ou seja, o Receptor e o Transmissor trocam de papéis.**
QUIT	**O Receptor envia um OK e então fecha o canal de comunicação com o Emissor-SMTP**

A figura 2.5 demonstra o funcionamento do protocolo SMTP.

FIGURA 2.5 – EXEMPLO DE FUNCIONAMENTO DO PROTOCOLO SMTP



O exemplo corresponde à transferência de uma mensagem do usuário silva* do domínio (especifica o equipamento em que reside a caixa postal do usuário) cyclades.com.br aos usuários mauro* e souza* do domínio netpro.com.br e a usuária claudia* do domínio dialdata.com.br, sendo que o usuário souza não tem caixa postal.

* Apesar de Claudia, Mauro, Silva e Souza serem substantivos próprios, foram colocados em letra minúscula porque são caixas postais de um domínio.

Inicialmente o cliente estabelece uma conexão com o servidor via TCP, e aguarda que o servidor envie uma mensagem de aceitando a conexão: “220 SMTP SERVER”.

Quando o cliente recebe a mensagem “220”, envia o comando “HELO”. Ao receber a mensagem “HELO”, o servidor envia outra com sua identificação. Após isto, o transmissor pode enviar suas mensagens. O receptor deve confirmar positiva ou negativamente a recepção de todas as mensagens. A mensagem positiva de recepção é: “250 OK”, caso haja algum problema e o transmissor não consiga acesso ao receptor, a mensagem SMTP será: “550 UNKNOWN USER NAME”. (Gasparini, 1994)

3 KIT DE DESENVOLVIMENTO RCM2200 (RABBIT CORE MODULE)

Para o desenvolvimento do protótipo foi utilizado o Kit de Desenvolvimento RCM2200, que inclui um módulo programável RCM2200 (com microprocessador Rabbit 2000™ com *clock* de 22.1 MHz, 256K de memória *flash*, 128K SRAM, interface Ethernet e portas seriais, além de entradas e saídas digitais), placa protótipo, ambiente de desenvolvimento Dynamic C com bibliotecas para TCP/IP, um cabo serial para programação, fonte de alimentação de 12V, manual de iniciação e documentação em CD-ROM. A figura 3.1 mostra o kit de desenvolvimento RCM2200.

FIGURA 3.1 – KIT DE DESENVOLVIMENTO RCM2200

3.1 – MÓDULO RCM2200

O módulo RCM2200 é equipado com uma interface *Ethernet* 10Base-T (conector RJ-45), 256k de memória *flash* e 128K de memória estática. Este módulo suporta aplicações que utilizam protocolos da arquitetura TCP/IP. O módulo RCM2200 é ilustrado nas figuras 3.2(vista superior) e 3.3 (vista inferior) em tamanho real.

FIGURA 3.2 – MÓDULO RCM2200 (VISTA SUPERIOR)

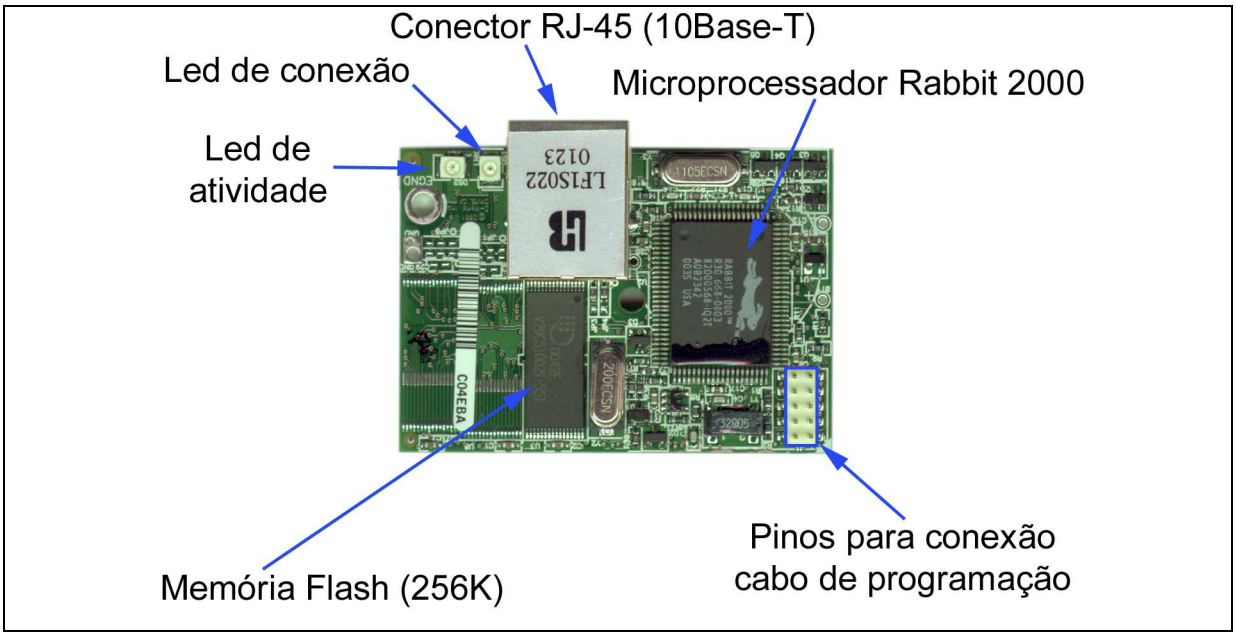
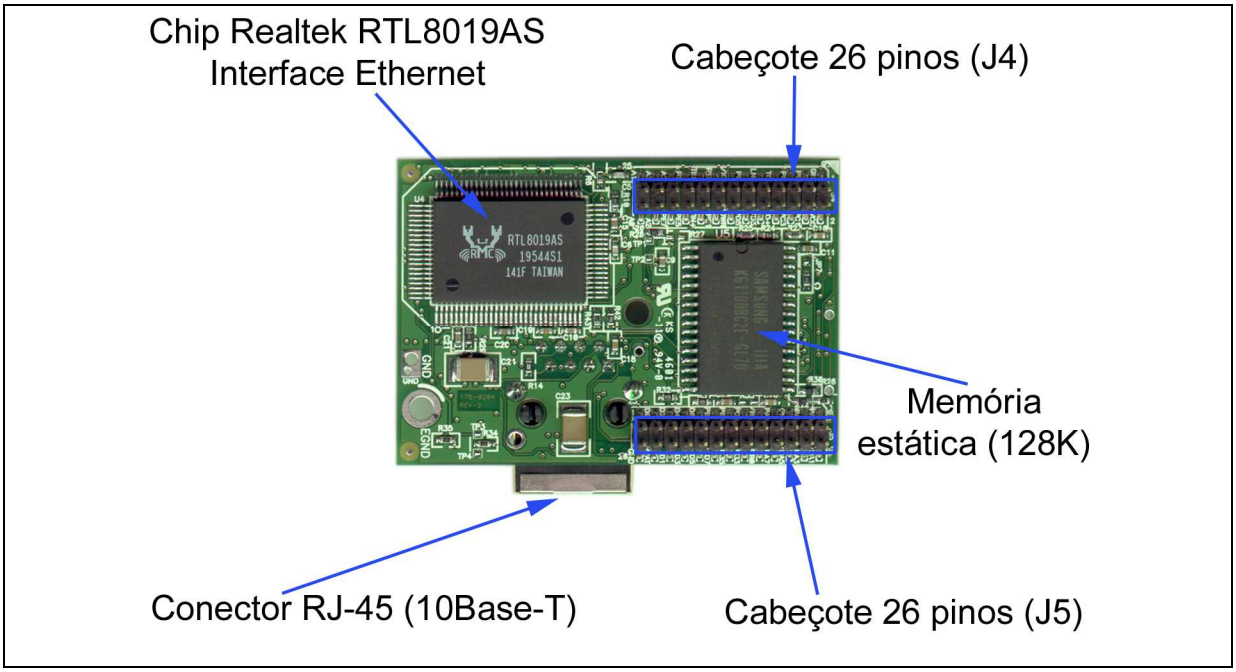


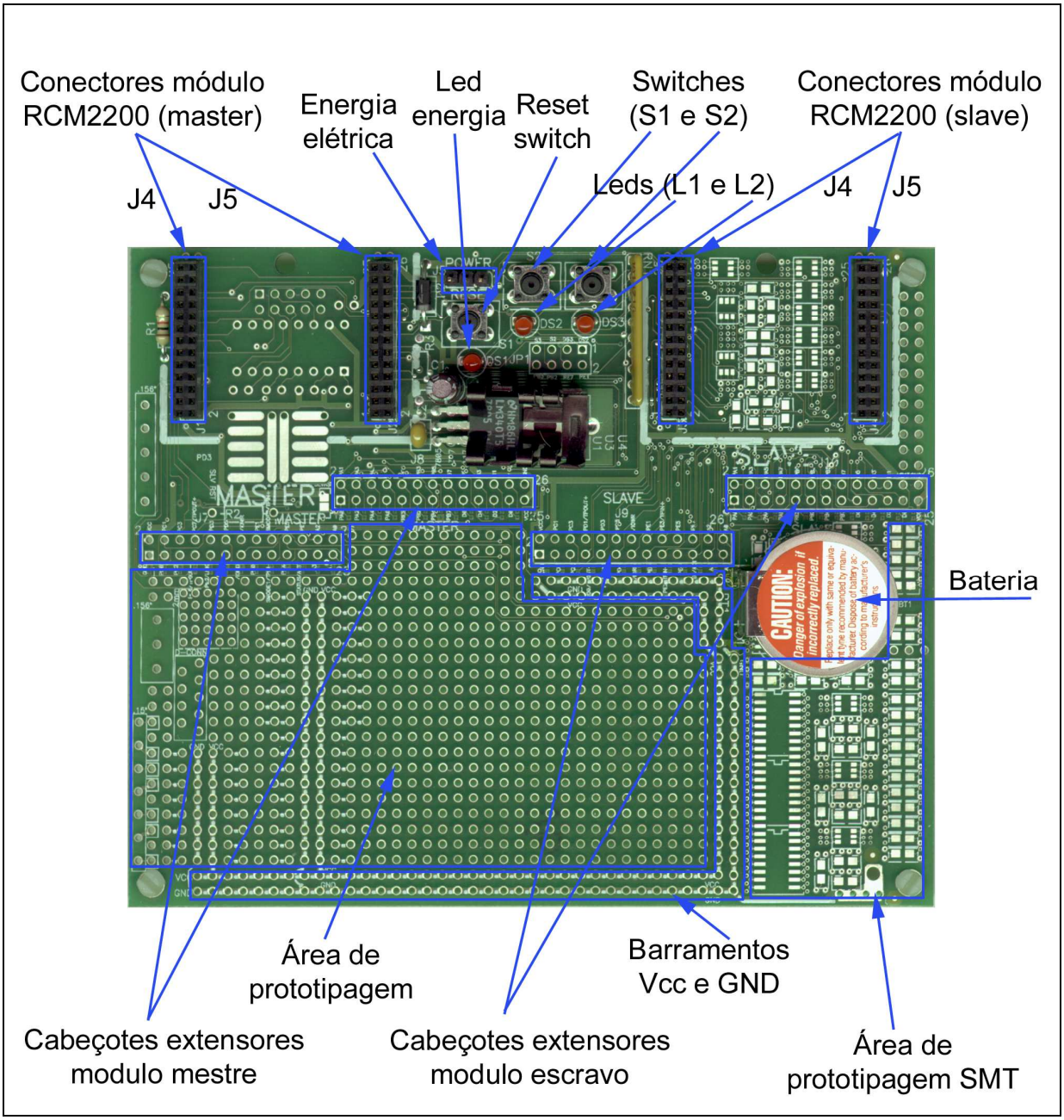
FIGURA 3.3 - MÓDULO RCM2200 (VISTA INFERIOR)



3.2 – PLACA PROTÓTIPO

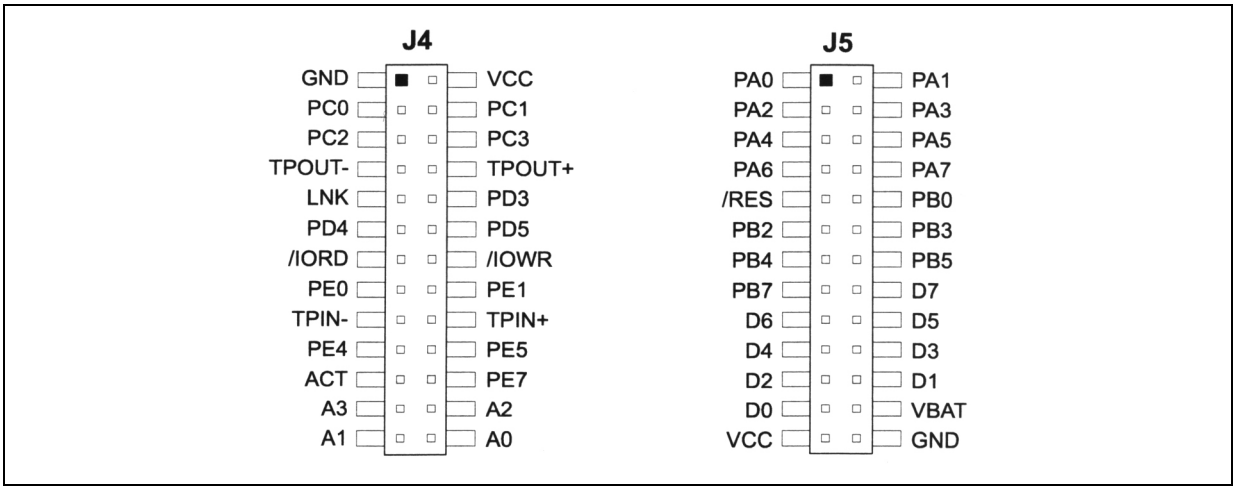
A placa protótipo incluída no Kit de Desenvolvimento torna fácil a conexão do módulo RCM2200 a uma fonte elétrica, além de prover algumas entradas/saídas básicas (*switches* e *leds*) assim como uma área de protótipo para desenvolvimento de *hardwares* mais avançados. A figura 3.4 ilustra a placa protótipo em tamanho real com suas principais ferramentas.

FIGURA 3.4 – PLACA PROTÓTIPO DO KIT DE DESENVOLVIMENTO



Para conexão com a placa protótipo os módulos RCM2200 utilizam os cabeçotes J4 e J5, ilustrados na figura 3.3. A figura 3.5 mostra as funções dos pinos dos cabeçotes J4 e J5.

FIGURA 3.5 – FUNÇÕES DOS PINOS DOS CABEÇOTES J4 E J5 DO RCM2200



3.3 – AMBIENTE DE DESENVOLVIMENTO DYNAMIC C

O ambiente de programação é o Dynamic C, desenvolvido pela Z-World. Este provê completa compatibilidade com os protocolos ICMP (Internet Control Message Protocol), HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), POP (Post Office Protocol), FTP (File Transfer Protocol) e TFTP (Trivial File Transfer Protocol) (cliente e servidor), permitindo às aplicações escreverem diretamente nos *sockets* TCP e UDP. O Dynamic C inclui um editor, compilador, depurador e bibliotecas de funções. A tabela 3.1 mostra algumas destas bibliotecas.

TABELA 3.1 – BIBLIOTECAS DO DYNAMIC C

costate.lib	funções para gerenciamento multitarefa
math.lib	funções matemáticas
rs232.lib	funções para transferência de dados
rtclock.lib	funções para controle de tempo
string.lib	funções para tratamento de caracteres
xmem	funções para controle de acesso a memória

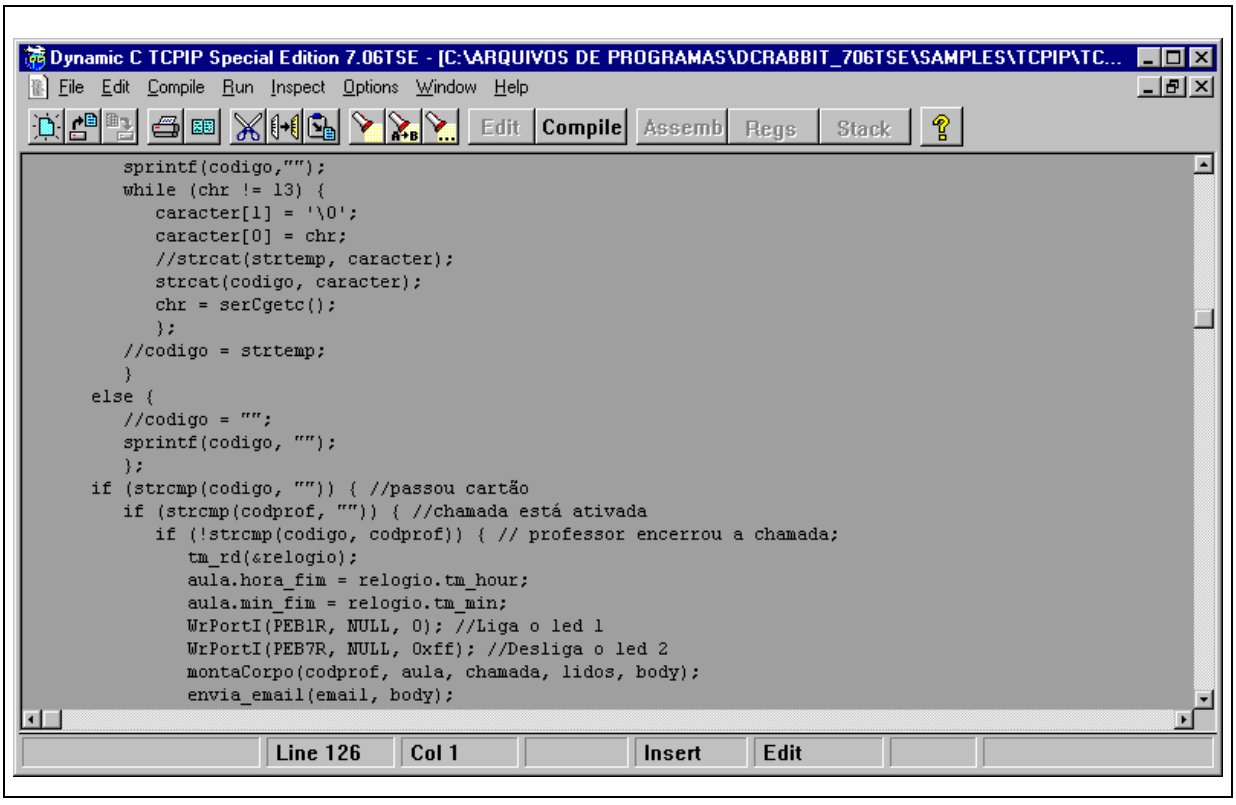
A tabela 3.2 ilustra bibliotecas para uso exclusivo com o protocolo TCP/IP.

TABELA 3.2 – BIBLIOTECAS DO DYNAMIC C PARA USO EXCLUSIVO COM O PROTOCOLO TCP/IP

arp.lib	funções para resolução de endereços IP
bootp.lib	funções para protocolo *bootstrap*
drctcp.lib	funções para os protocolos TCP e UDP
ftp_client.lib	funções para protocolo FTP (aplicações cliente)
ftp_server.lib	funções para protocolo FTP (aplicações servidor)
http.lib	funções para protocolo HTTP
icmp.lib	funções para protocolo ICMP
pktdrv.lib	funções para *packetdriver*
pop3.lib	funções para protocolo POP3
smtp.lib	funções para protocolo SMTP

O ambiente de programação Dynamic C é ilustrado na figura 3.6.

FIGURA 3.6 – AMBIENTE DE PROGRAMAÇÃO DYNAMIC C



4 ESPECIFICAÇÃO

4.1 – PROTÓTIPO

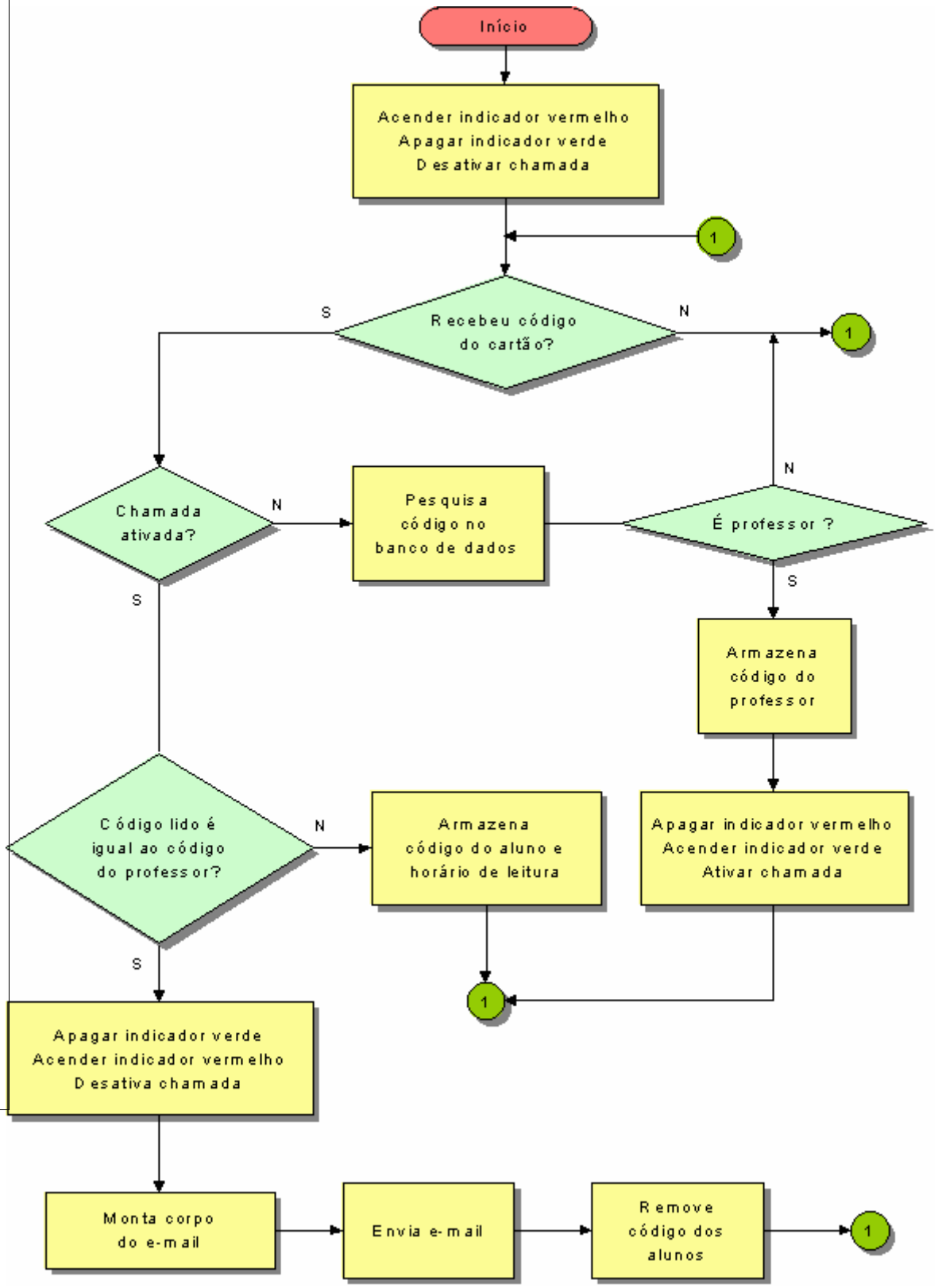
O protótipo deve executar as seguintes tarefas: ler os cartões de alunos e professores, armazenando-os na memória juntamente com seu horário de leitura, montagem do corpo do *e-mail* na memória e envio do mesmo para o endereço eletrônico do professor.

O professor, ao “passar” seu cartão, ativa a recepção dos cartões dos alunos (inicia a chamada), a partir desse momento todos os códigos dos cartões lidos são armazenados, até que o professor desative a recepção dos cartões dos alunos (encerra a chamada). O protótipo possui dois indicadores (luzes), um verde e um vermelho, o vermelho aceso indica que a leitura de cartões está desabilitada, enquanto que o verde indica que a leitura de cartões está habilitada.

Quando a chamada for encerrada, o protótipo deve montar o corpo do *e-mail*. Para tal, ele deve estabelecer uma conexão com um aplicativo servidor (pelo protocolo TCP) para buscar informações (nome e endereço eletrônico do professor, nomes dos alunos, nome da cadeira e do curso, etc) do banco de dados. O acesso ao banco de dados é feito pelo aplicativo servidor. Ao final deste processo, um *e-mail* contendo o corpo montado é enviado ao endereço eletrônico do professor.

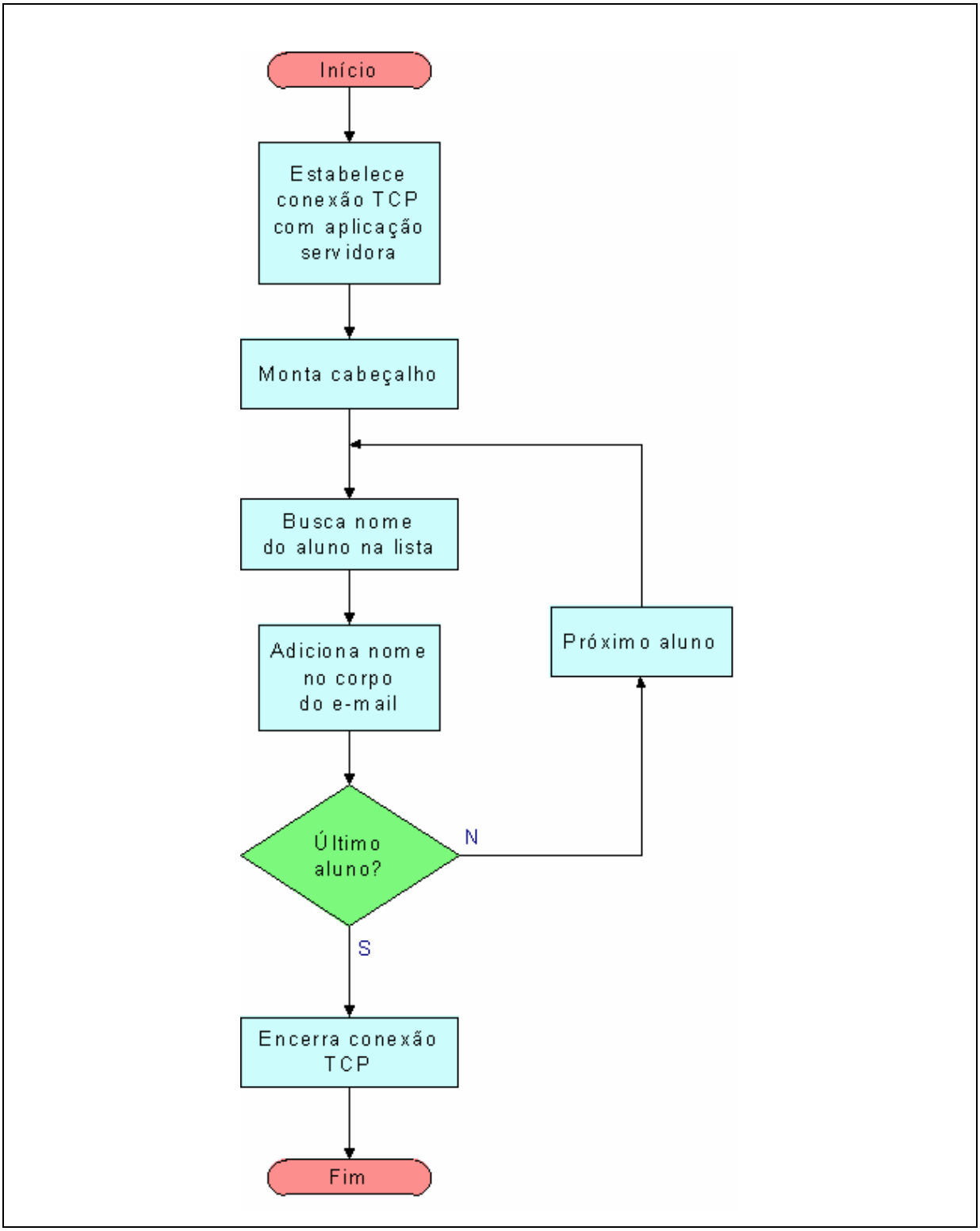
O fluxograma 4.1 proporciona uma visão geral da aplicação cliente.

FLUXOGRAMA 4.1 – VISÃO GERAL DA APLICAÇÃO CLIENTE



O fluxograma 4.2 mostra o processo de montagem do corpo do *e-mail*.

FLUXOGRAMA 4.2 – MONTAGEM DO CORPO DO *E-MAIL*

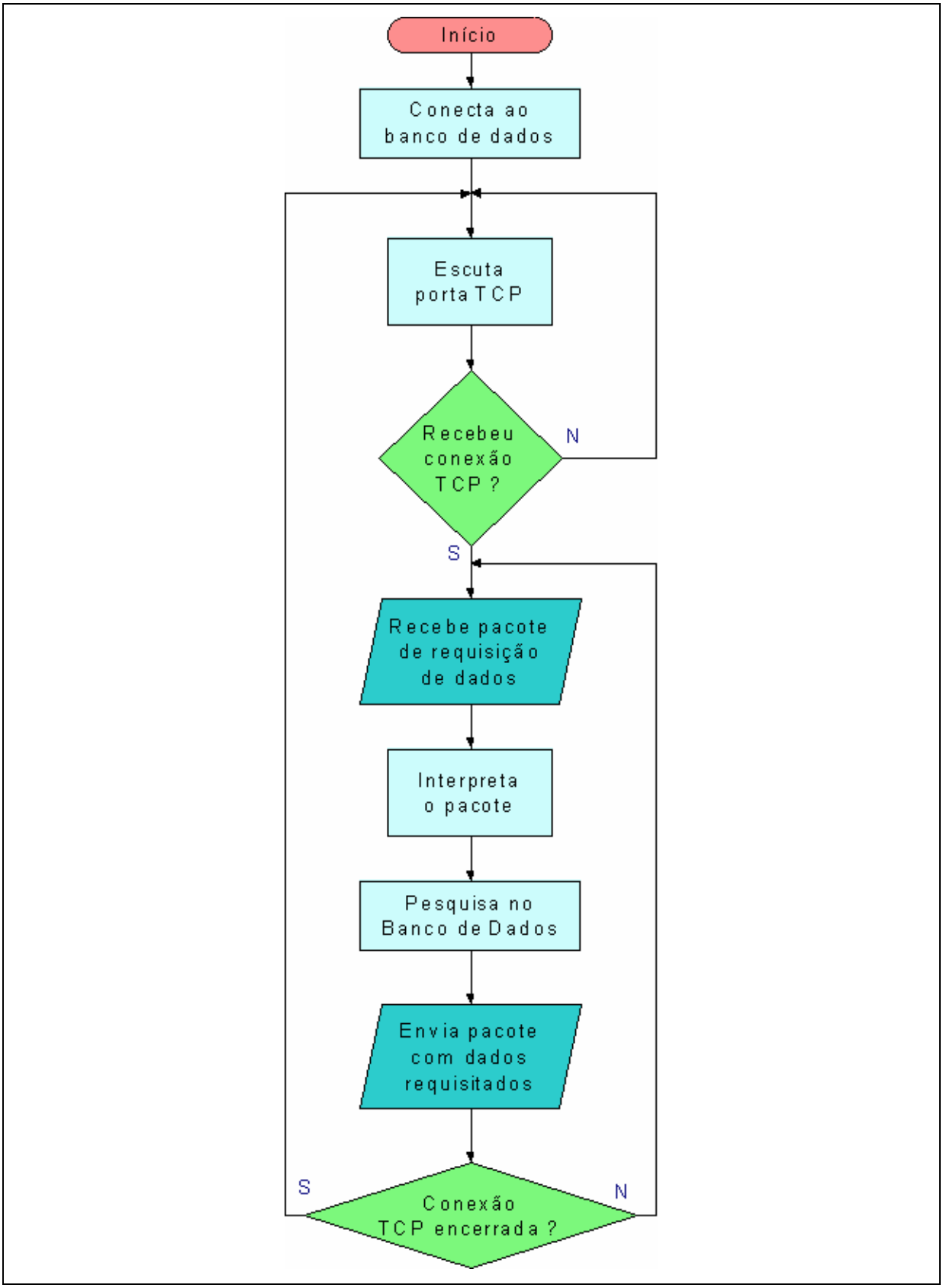


O aplicativo servidor tem como principal objetivo receber requisições de informações das aplicações cliente, pesquisar estas informações na base de dados e finalmente, enviar os resultados à aplicação cliente requisitante.

Para tanto, ela fica “escutando” uma porta (*port*) específica no computador onde está o banco de dados da universidade. A cada conexão TCP recebida, ela recebe o pacote de requisição de dados. Interpreta o mesmo com a finalidade de definir a pesquisa a ser feita. Faz a pesquisa na base de dados e envia um pacote com os dados requisitados à aplicação cliente que os requisitou.

O fluxograma 4.3 mostra uma visão geral do aplicativo servidor.

FLUXOGRAMA 4.3 – VISÃO GERAL DO APLICATIVO SERVIDOR



4.2 – COMUNICAÇÃO CLIENTE / SERVIDOR (MENSAGENS)

Durante a conexão com a aplicação servidora, serão efetuadas trocas de mensagens. Cada mensagem enviada à aplicação servidora será uma função, esperando portanto um resultado para a mesma. Tal resultado será enviado pela aplicação servidora em forma de mensagem. Os parâmetros serão separados pelo caractere especial "|" (*pipe*). Abaixo estão as funções especificadas:

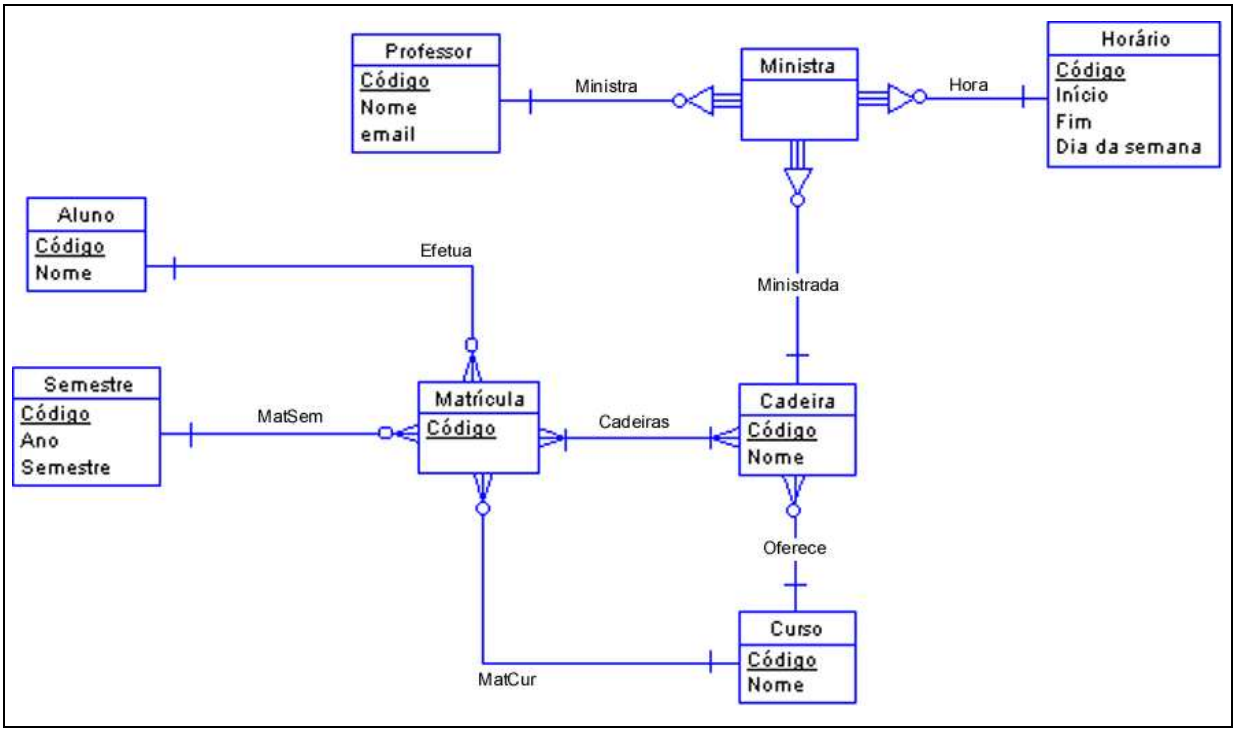
1. EHPROFESSOR – Esta função verifica na base de dados se o código fornecido pertence a um professor. Se pertencer o resultado será o endereço eletrônico do mesmo, senão será a frase "NÃO É PROFESSOR". Sintaxe: EHPROFESSOR|[código].
2. NOMEPROFESSOR – Busca na base de dados o nome do professor de acordo com o código fornecido, e retorna o mesmo como resultado. Sintaxe: NOMEPROFESSOR|[código].
3. NOMEALUNO – Procura pelo nome do aluno na base de dados, conforme código fornecido. Se o nome for encontrado o resultado será o mesmo, senão será a frase "NÃO É ALUNO".
4. CODIGOCADEIRA – Busca na base de dados o código da cadeira ministrada por determinado professor, em determinado horário e dia da semana. O resultado é o código obtido. Sintaxe: CODIGOCADEIRA|[horário]|[dia da semana]|[código do professor].
5. CODIGOCURSOCADEIRA – Tem como resultado o código do curso de determinada cadeira encontrado na base de dados. Sintaxe: CODIGOCURSOCADEIRA|[código da cadeira].
6. NOMECURSO – Busca na base de dados o nome do curso, conforme o código fornecido. O resultado é o nome encontrado. Sintaxe: NOMECURSO|[código].
7. NOMECADEIRA – Tem como resultado o nome de determinada cadeira encontrada na base de dados. Sintaxe: NOMECADEIRA|[código].

8. MATRICULADO – Esta função verifica se determinado aluno está matriculado em determinada cadeira, de determinado curso em determinado semestre. O resultado será a palavra “MATRICULADO” em caso verdadeiro e a frase “NÃOMATRICULADO” em caso contrário. Sintaxe: MATRICULADO|[código do aluno]|[código da cadeira]|[código do curso]|[ano]|[mês].

4.3 – MODELAGEM DE DADOS

A aplicação servidora será programada para efetuar pesquisas numa determinada modelagem de dados. Esta modelagem é descrita na figura 4.1.

FIGURA 4.1 – MODELAGEM DA BASE DE DADOS



5 IMPLEMENTAÇÃO

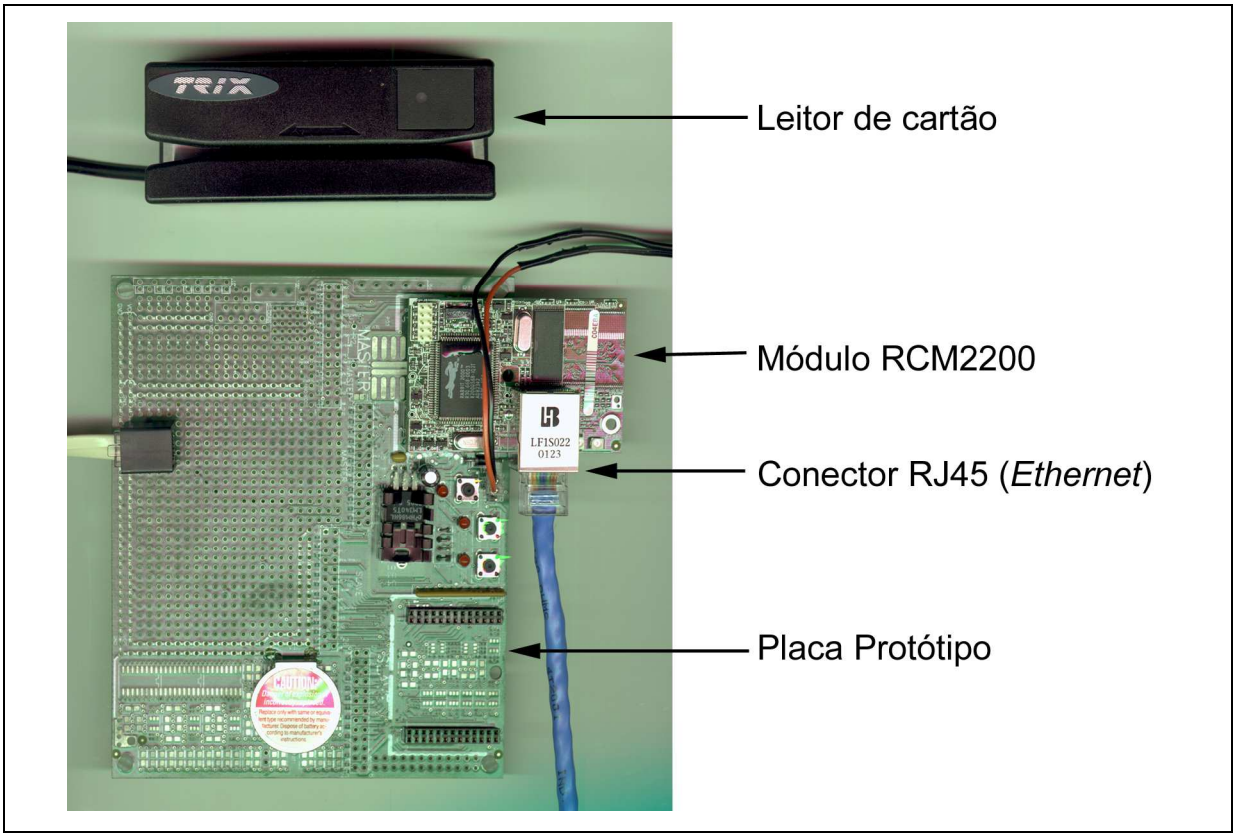
5.1 HARDWARE

O sistema é composto de placa programável (*kit* RCM 2200), placa protótipo, leitor de cartão (código de barras) e fonte de alimentação.

A placa programável conectada a placa protótipo, é responsável pela interpretação dos **sinais** provenientes do leitor de cartão, comunicação com uma aplicação servidora e pela montagem e envio do *e-mail*.

O leitor de cartão interpreta o código de barras existente no cartão e transmite usando o padrão de sinais RS232.

FIGURA 5.1 – HARDWARE DO SISTEMA



5.2 SOFTWARE

Foram desenvolvidos dois softwares distintos. O primeiro é uma aplicação servidora, a qual fica “escutando” uma porta (*port*) específica no computador onde está o banco de dados da universidade. Esta aplicação interage com o banco de dados, obtendo as informações requisitadas.

O segundo software é o armazenado na placa programável (aplicação cliente). Este software se conecta com a aplicação servidora, fazendo requisições à mesma, de acordo com sua programação. Ele é responsável por administrar os códigos dos cartões lidos, montar o *e-mail* contendo os nomes dos alunos e professor, e enviar o mesmo ao professor que ministra determinada aula.

5.2.1 APLICAÇÃO SERVIDORA

Esta aplicação foi desenvolvida na linguagem de desenvolvimento Borland Delphi 5.0, e suporta conexão ao banco de dados Interbase, versão 1.00 distribuído pela empresa Inprise Corporation. Para efetuar pesquisas na base de dados, esta aplicação utiliza a linguagem SQL (*Structured Query Language*) para comunicação com o banco de dados.

Foi utilizado no desenvolvimento o componente **TsocketServer**, fornecido juntamente com a linguagem Borland Delphi 5.0. Este componente tem por característica principal “escutar” uma determinada porta (*port*), aguardando por conexões TCP (*Sockets*) clientes.

A porta digital deve ser informada antes da ativação da aplicação. Este valor deve ser conhecido, pois é através dele, juntamente com o endereço IP da máquina onde a aplicação servidora está sendo executada, que a aplicação cliente fará conexão, através dos *sockets* TCP, conforme visto anteriormente.

Ao receber uma mensagem de um cliente, no formato de texto, este componente gera um evento, chamado **OnClientRead**. Neste evento é possível obter informações como texto recebido, endereço IP e porta lógica do cliente.

Quando este evento é gerado, a mensagem recebida é interpretada, então esta aplicação efetua uma pesquisa no banco de dados, e retorna o resultado para a aplicação cliente que enviou a mensagem.

O quadro demonstra como a mensagem recebida é interpretada no evento **OnClientRead.**

QUADRO 5.1 - INTERPRETAÇÃO DAS MENSAGENS NO EVENTO ONCLIENTREAD

```
procedure TServerFrm.ServerSocketClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
var
  s, email, sRecebido, sRemoteIP : string;
  horario, dsemana, codProf, cadeira, cod : string;
  curso, ano, mes : integer;
begin
  sRecebido := Socket.ReceiveText;
  sRemoteIP := Socket.RemoteAddress;
  Memo.Lines.Add('Recebido '+sRemoteIP+' -> '+sRecebido);
  s := GetShortHint(sRecebido);
  if s = 'EHPROFESSOR' then begin
    s := GetLongHint(sRecebido);
    cod := s;
    if EhProfessor(cod, email) then
      Socket.SendText(email)
    else
      Socket.SendText('NAOEHPROFESSOR');
    end
  else if s = 'NOMEPROFESSOR' then begin
    s := GetLongHint(sRecebido);
    cod := s;
    Socket.SendText(NomeProfessor(cod));
    end
  else if s = 'NOMEALUNO' then begin
    s := GetLongHint(sRecebido);
    socket.SendText(NomeAluno(s));
```

QUADRO 5.1 - INTERPRETAÇÃO DAS MENSAGENS NO EVENTO ONCLIENTREAD - CONTINUAÇÃO

```
            end
          else if s = 'CODIGOCADEIRA' then begin
            s := GetLongHint(sRecebido);
            horario := GetShortHint(s);
            s := GetLongHint(s);
            dsemana := GetShortHint(s);
            codProf := GetLongHint(s);
            socket.SendText(CodigoCadeira(horario, dsemana, codProf));
            end
          else if s = 'CODIGOCURSOCADEIRA' then begin
            s := GetLongHint(sRecebido);
            Socket.SendText(CodigoCursoCadeira(s));
            end
          else if s = 'NOMECURSO' then begin
            s := GetLongHint(sRecebido);
            Socket.SendText(NomeCurso(s));
            end
          else if s = 'NOMECADEIRA' then begin
            s := GetLongHint(sRecebido);
            Socket.SendText(NomeCadeira(s));
            end
          else if s = 'MATRICULADO' then begin
            s := GetLongHint(sRecebido);
            cod := GetShortHint(s);
            s := GetLongHint(s);
            cadeira := GetShortHint(s);
            s := GetLongHint(s);
            curso := strtoint(GetShortHint(s));
            s := GetLongHint(s);
            ano := strtoint(GetShortHint(s));
            s := GetLongHint(s);
            mes := strtoint(s);
            if Matriculado(cadeira, cod, curso, ano, mes) then
              socket.sendtext('MATRICULADO')
            else
              socket.sendtext('NAOMATRICULADO');
            end;

end;
```

De acordo com a mensagem interpretada, esta aplicação executará um determinado comando SQL, abaixo temos alguns exemplos.

Se a mensagem recebida for “NOMEPROFESSOR”, o comando SQL executado será:

“SELECT PR_NOME
FROM PROFESSOR
WHERE
PR_CODIGO = {código do professor}”.

Se a mensagem interpretada for “CODIGOCADEIRA”, o comando SQL executado será:

“SELECT CA_CODIGO
FROM CADEIRA, MINISTRA, HORARIO
WHERE
CADEIRA.CA_CODIGO = MINISTRA.CADEIRA_CODIGO AND
MINISTRA.HORARIO_CODIGO = HORARIO.HO_CODIGO AND
MINISTRA.PROFESSOR_CODIGO = {código do professor} AND
HORARIO.HO_INICIO <= {horário} AND
HORARIO.HO_FIM >= {horário} AND
HO_DIASEMANA = {dia da semana}”.

5.2.2 APLICAÇÃO CLIENTE

Para o desenvolvimento da aplicação cliente, foram utilizadas bibliotecas de funções fornecidas pela Z_World juntamente com a linguagem de desenvolvimento Dynamic C.

Para que esta aplicação possa funcionar de maneira adequada é necessário inicializar algumas constantes como: endereço IP, máscara de rede, *gateway* (equipamento que permite a interconexão de redes compatíveis ou não), endereço IP do servidor de nomes (DNS), endereço IP do servidor de *e-mails* (SMTP) e o endereço

(IP e *port*) da aplicação servidora. Detalhes sobre o significado destes termos podem ser encontrados em Carvalho(1994).

O quadro 5.2 demonstra como estas constantes foram definidas na aplicação.

QUADRO 5.2 – CONSTANTES DA APLICAÇÃO CLIENTE

#define MY_IP_ADDRESS	"200.135.24.126"	//Endereço IP local
#define MY_NETMASK	"255.255.255.0"	//Máscara de rede
#define MY_GATEWAY	"200.135.24.40"	//*Gateway*
#define MY_NAMESERVER	"200.135.24.7"	//Servidor DNS
#define SMTP_SERVER	"200.135.24.2"	//Servidor SMTP
//*Socket* aplicação servidora		
#define REMOTE_IP	"200.135.24.67"	//Endereço IP remoto
#define REMOTE_PORT	30150	//Porta remota (*Port*)

5.2.2.1 MÓDULO DE ENVIO DE E-MAIL

Para envio do *e-mail* com a lista de freqüência de determinada aula, foi utilizada a biblioteca de funções SMTP.LIB. São necessárias três funções desta biblioteca, que devem ser executadas na ordem descrita abaixo:

a) smtp_sendmail(REMETENTE, DESTINATARIO, ASSUNTO, CORPO);
b) smtp_mailtick();
c) smtp_status().

A função **smtp_sendmail** efetua o envio do e-mail, utilizando os parâmetros REMETENTE, DESTINATARIO, ASSUNTO, CORPO que significam respectivamente: endereços eletrônicos do remetente e destinatário da mensagem, assunto da mensagem e o corpo da mesma.

A função **smtp_mailtick** retorna o estado do envio do *e-mail*. Estes estados podem ser:

a) SMTP_SUCCESS – *e-mail* enviado com sucesso;
b) SMTP_PENDING – *e-mail* ainda não foi enviado por completo;
c) SMTP_TIME – terminou o tempo para envio do *e-mail*;
d) SMTP_UNEXPECTED – resposta inválida recebida do servidor SMTP.

A função **smtp_status** retorna o estado do último *e-mail* processado. Estes estados são os mesmo da função **smtp_mailtick**.

O quadro 5.3 demonstra como estas funções foram utilizadas na aplicação.

QUADRO 5.3 – IMPLEMENTAÇÃO DO ENVIO DE *E-MAIL*

```
smtp_sendmail(to, from, subject, body);

while(smtp_mailtick()==SMTP_PENDING)
        continue;
```

5.2.2.2 MÓDULO DE MONTAGEM DO CORPO DO E-MAIL

Este módulo é responsável pela montagem do corpo do *e-mail* em formato texto. Para tal, ele deve estabelecer uma conexão com a aplicação servidora, buscando informações como: nome do curso, nome da disciplina, nome e endereço eletrônico do professor, nome do aluno, etc.

Para obter estas informações, este módulo envia mensagens específicas à aplicação servidora, dependendo da informação necessária. Estas mensagens, conforme definidas no capítulo de especificação, são interpretadas pela aplicação servidora, identificando uma determinada função e retornando um resultado. O módulo então aguarda este resultado para adicionar ao corpo do *e-mail*.

O quadro 5.4 demonstra uma parte da função de montagem do corpo do *e-mail*, onde são enviadas mensagens à aplicação servidora.

QUADRO 5.4 – PARTE DA FUNÇÃO DE MONTAGEM DO CORPO DO *E-MAIL*

```
//busca codigo/nome da cadeira e codigo/nome do curso
//codigo cadeira
sprintf(buffer, "CODIGOCADEIRA|");
strcat(buffer, horario);
strcat(buffer, "|");
strcat(buffer, dsemana);
strcat(buffer, "|");
strcat(buffer, codProf);
costate{
        waitfor(sock_puts(&s,buffer));
        };
sock_wait_input(&s,0,NULL,&status);
if(sock_gets(&s,buffer,30)) {
        sprintf(cod_cadeira, buffer);
        };

//codigo do curso daquela cadeira
sprintf(buffer,"CODIGOCURSOCADEIRA|");
strcat(buffer, cod_cadeira);
costate{
        waitfor(sock_puts(&s,buffer));
        };
sock_wait_input(&s,0,NULL,&status);
if(sock_gets(&s,buffer,30)) {
        sprintf(cod_curso, buffer);
        };

//nome do curso
sprintf(buffer, "NOMECURSO|");
strcat(buffer, cod_curso);
costate{
        waitfor(sock_puts(&s,buffer));
        };
sock_wait_input(&s,0,NULL,&status);
if(sock_gets(&s,buffer,30)) {
        strcat(corpo, "Curso: ");
        strcat(corpo, buffer);
        strcat(corpo, "\n");
        };

//nome cadeira
sprintf(buffer, "NOMECADEIRA|");
strcat(buffer, cod_cadeira);
costate{
        waitfor(sock_puts(&s,buffer));
```

QUADRO 5.4 – PARTE DA FUNÇÃO DE MONTAGEM DO CORPO DO *E-MAIL* - CONTINUAÇÃO

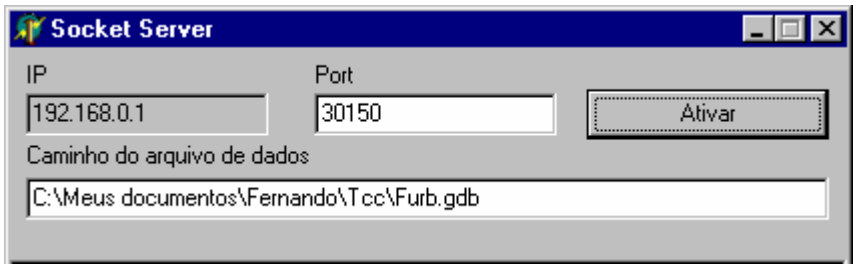
```
            };
        sock_wait_input(&s,0,NULL,&status);
        if(sock_gets(&s,buffer,30)) {
            strcat(corpo, "Cadeira: ");
            strcat(corpo, buffer);
            strcat(corpo, "\n");

            };
```

5.3 OPERACIONALIDADE DA IMPLEMENTAÇÃO

O primeiro passo para inicializar o sistema, é a ativação da aplicação servidora. Para tal deve-se indicar o caminho do arquivo de dados, assim como a porta (*port*) por onde as aplicações clientes se conectarão. O campo IP indica o número IP do computador onde a aplicação está sendo executada. A figura 5.2 a aplicação servidora antes da ativação.

FIGURA 5.2 – APLICAÇÃO SERVIDORA

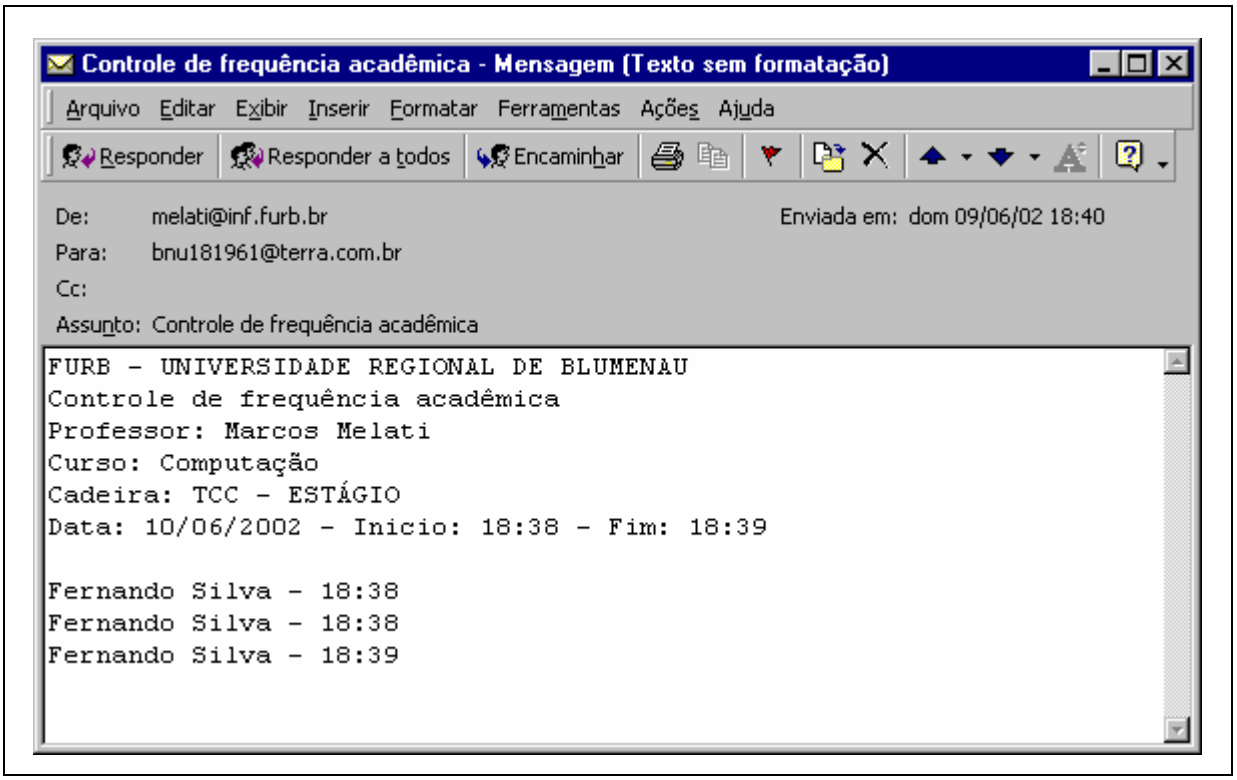


Estando o protótipo corretamente conectado à rede da universidade, basta que sejam “passados” os cartões no leitor de cartões para que o sistema faça o seu processamento. Quando um cartão com o código de um professor for lido, a lista de freqüência é ativada, o indicativo para tal é o *led*. Todos os códigos dos cartões lidos serão armazenados juntamente com o horário de leitura, até que o cartão com o código do professor seja lido novamente. Indicando o encerramento da lista de freqüência.

É neste momento que o *e-mail* será montado de acordo com as informações encontradas na base de dados e enviado ao professor.

A figura 5.3 demonstra um *e-mail* montado e enviado pelo protótipo.

FIGURA 5.3 – *E-MAIL* MONTADO E ENVIADO PELO PROTÓTIPO



6 CONCLUSÕES

Verificou-se que o objetivo do trabalho proposto foi alcançado. Para tal, foi buscado um aprendizado conceitual e tecnológico dos estudos descritos neste trabalho. Verificou-se também, que é possível fazer a automação do processo de freqüência acadêmica, a um custo acessível para uma instituição de ensino.

As ferramentas (software e hardware) utilizadas para implementação, mostraram-se adequadas e eficientes, possibilitando futuras implementações ou manutenções de forma prática.

Foram encontradas algumas dificuldades no decorrer deste trabalho, principalmente com relação à eletrônica, leitura e escrita das portas na placa programável. Estas dificuldades foram superadas com o auxílio do professor orientador e das fontes de pesquisa. Outras dificuldades foram encontradas na compra dos equipamentos eletrônicos, devido ao seu alto custo. Com o auxílio da FURB, que nos emprestou um leitor de cartão, não se fez necessária a compra do mesmo, o que ajudou a reduzir o investimento financeiro do aluno.

Devido ao fato de terem sido desenvolvidas duas aplicações simultaneamente, algumas dificuldades surgiram na adaptação à linguagem de programação C. A programação simultânea em duas linguagens prejudicou na sintaxe durante o desenvolvimento das aplicações. Nestes momentos a experiência e conhecimentos do professor orientador foram muito importantes.

Dentro do objetivo proposto, encontramos uma limitação no que diz respeito à aplicação servidora. Ela é programada para funcionar com uma determinada modelagem de dados, ou seja, para cada modelagem de dados, deve ser programada uma nova aplicação servidora. No entanto, em futuras implementações podem ser adicionados diversos novos recursos, como uma aplicação servidora configurável à modelagem de dados, uma página de pesquisas às listas de freqüências pela Internet ou uma configuração automática das aplicações cliente através da aplicação servidora.

6.1 EXTENSÕES

Como sugestões para continuidades deste trabalho pode-se citar:

a) rotinas de configuração automática das aplicações clientes através da aplicação servidora, configurando automaticamente o endereço IP, horário;
b) implementação de uma página na Internet para pesquisa às listas de freqüências;
c) implementação da aplicação servidora tornando-a compatível com outros bancos de dados e configurável a diversas modelagens de dados;
d) incluir criptografia aos *e-mails*.

REFERÊNCIAS BIBLIOGRÁFICAS

MARCOVITCH, J. **A universidade (im)possível.** São Paulo: Futura, 1998.

FURB, UNIVERSIDADE REGIONAL DE BLUMENAU. **Regimento geral da Universidade Regional de Blumenau**, Blumenau, dez. 1995. Disponível em: <http://www.furb.rct-sc.br/sinsepes/regimento_universidade.htm>. Acesso em: 30 maio 2001.

GASPARINI, A. F. L.; BARRELA, F. E. **TCP/IP solução para conectividade**. São Paulo: Érica, 1993.

TANENBAUM, A. S. **Redes de computadores**. Rio de Janeiro: Campus, 1994.

SOARES, L. F. G.; COLCHER, S.; LEMOS, G. **Rede de computadores** - das LANs, MANs e WANs às redes ATM. Rio de Janeiro: Campus, 1995.

RABBIT. **RabbitCore RCM2200 –** Getting started manual. Califórnia: Rabbit Semiconductor, 2000.

SCHILDT, H. **C, completo e total**. São Paulo: Makron Books, 1991.

DUMAS, Arthur. **Programando WinSock**. Rio de Janeiro: Axcel Books, 1995.

WERNER, José Alberto Vasi. **Internet e arquitetura TCP/IP.** Rio de Janeiro: PUC-Rio, 1999.

CHIOZZOTTO, Mauro; SILVA, Luis Antonio Pinto da. **TCP-IP** : tecnologia e implementação. São Paulo: Érica, 1999.

CARVALHO, Tereza Cristina Melo de Brito. **Arquiteturas de redes de computadores OSI e TCP/IP**. Brasilia: SGA, 1994.