Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

Detalhes de desenvolvimeto do sistema gerador de Stubs - GAPS

HELI HENRIQUES ALCANTARA NASCIMENTO
MARCOS JOSÉ SANTANA
NIVALDI CALÔNEGO JUNIOR
REGINA HELENA CARLUCCI SANTANA

No. 37

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos
Nov./1995

Agradecimentos

À CAPES, à FAPESP e ao CNPq, pelo apoio financeiro.

Índice

Lista de Figuras

RESUMO

Este relatório apresenta detalhes da especificação e da implementação do sistema GAPS (Gerador Automático de Procedimentos *stubs*), para utilização no ambiente computacional do Laboratório de Sistemas Digitais do ICMSC/USP, em plataformas DOS. O gerador de *stubs* é uma ferramenta de auxílio ao desenvolvimento de aplicações distribuídas e tem como função criar automaticamente os procedimentos *stubs*, a partir da definição dos serviços que serão executados remotamente. Esse *software* de apoio libera os programadores da implementação de rotinas que envolvam os protocolos básicos de comunicação.

Os procedimentos *stubs* são responsáveis pela comunicação entre os processos clientes e os serviços oferecidos. Com esses procedimentos, os processos cliente e servidor podem ser compilados e executados separadamente, em máquinas diferentes.

Dentre as vantagens do sistema gerador de *stubs*, a mais importante, e que está bem caracterizada neste trabalho, é o ganho de produtividade nos projetos de aplicações distribuídas, tornando extremamente atrativa sua adoção nesses projetos.

O sistema proposto está implementado na linguagem "C" e pode ser utilizado facilmente para a geração de aplicações distribuídas envolvendo equipamentos compatíveis com a linha IBM-PC, executando DOS versão 3 ou superior.

INTRODUÇÃO

Os sistemas computacionais (*hardware e software*) têm evoluído bastante, principalmente nesta última década. Este trabalho não foge a realidade e ataca uma das áreas em que o mercado vem investindo cada vez mais: o desenvolvimento de aplicações distribuídas, ou seja, o projeto de sistemas utilizando redes de computadores.

O auge hoje é a instalação de redes de computadores, a implantação de servidores de arquivos, servidores de impressão, conexão com a Internet, protocolos específicos (para transferência de arquivos, para correio eletrônico, para ligação remota, por exemplo), entre outros.

O sistema Gerador Automático de Procedimentos *Stubs* (GAPS), desenvolvido neste trabalho, é uma ferramenta de auxilio à implementação de aplicações distribuídas, com a qual tem-se um ganho extremamente grande de produtividade e facilidade no desenvolvimento.

O GAPS foi desenvolvido sobre os protocolos de comunicação implementados pelo grupo de Sistemas Distribuídos e Programação Concorrente do LaSD (Laboratório de Sistemas Digitais) do ICMSC (Instituto de Ciências Matemáticas de São Carlos).

O sistema computacional do LaSD é composto por uma rede de microcomputadores PCs sob o sistema operacional DOS e utiliza os protocolos definidos no *packet driver* de obtidos da Universidade de Clarkson.

Este relatório descreve os detalhes de desenvolvimento do GAPS e é composto pelas seguintes seções, além desta seção introdutória:

Na seção 1 são descritos alguns conceitos importantes para o entendimento do trabalho.

A seção 2 contém a especificação do projeto.

Os detalhes de implementação estão descritos na seção 3.

Na seção 4 são apresentadas as conclusões deste trabalho.

Finalizando, encontram-se a bibliografia e um apêndice.

SEÇÃO 1

CONCEITOS FUNDAMENTAIS

Nesta seção encontram-se conceitos indispensáveis à compreensão deste trabalho. São abordados o ambiente de desenvolvimento do sistema gerador (do LaSD), os protocolos utilizados e o conceitos de mecanismos de comunicação entre aplicações distribuídos. Além destes tópicos, a seção descreve os conceitos sobre sistema gerador de *stubs*.

1.1- Ambiente de Desenvolvimento

Topologia

O Sistema do LaSD é composto por microcomputadores (chamadas também de estações de trabalho) IBM-PC compatíveis, interligados numa topologia em barramento através através de placas ethernet e cabo coaxial (figura 1.1). Em cada micro está instalado o sistema operacional DOS e o *packet driver* (NE1000 ou NE2000).

Além dos PCs, existe o Servidor de Processamento Paralelo projetado pela equipe do laboratório. Esse servidor está conectado à rede do laboratório via um PC que atua como processador de entrada.

Sistema Distribuído

Com relação ao *software*, foram implementados vários sistemas com base no modelo cliente/servidor. Entre outros, já foram desenvolvidos, no laboratório, os servidores de arquivos, de nomes, de impressão, e de processamento paralelo.

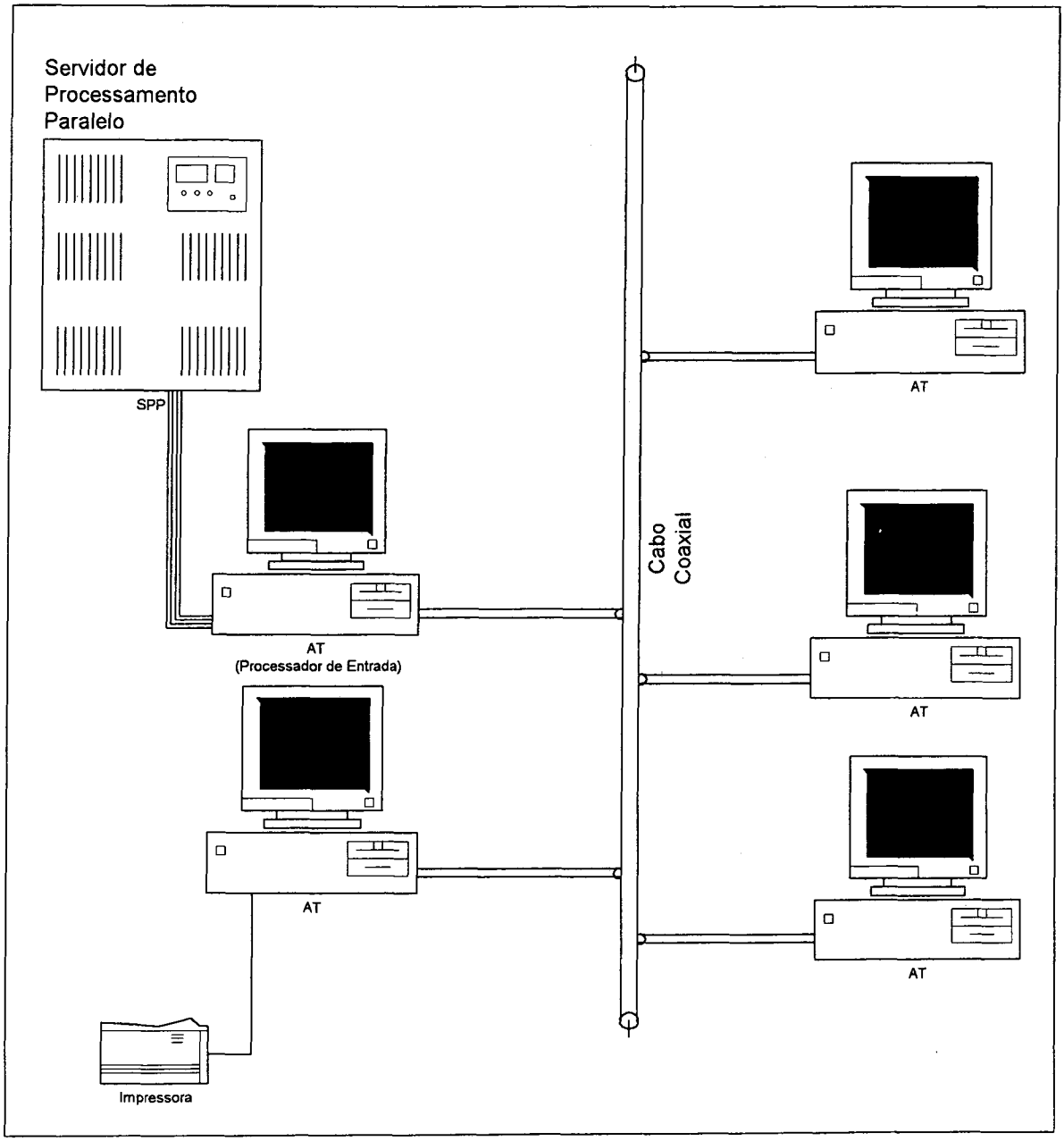

Figura 1.1- **Topologia da Rede de Computadores do LaSD.**

Arquitetura

Os protocolos de comunicação, desenvolvidos pelo LaSD foram implementados em turbo-C e utilizam os serviços do *packet driver*.

Na figura 1.2 são apresentadas as camadas e os respectivos protocolos do subsistema de interconexão do sistema computacional do LaSD.

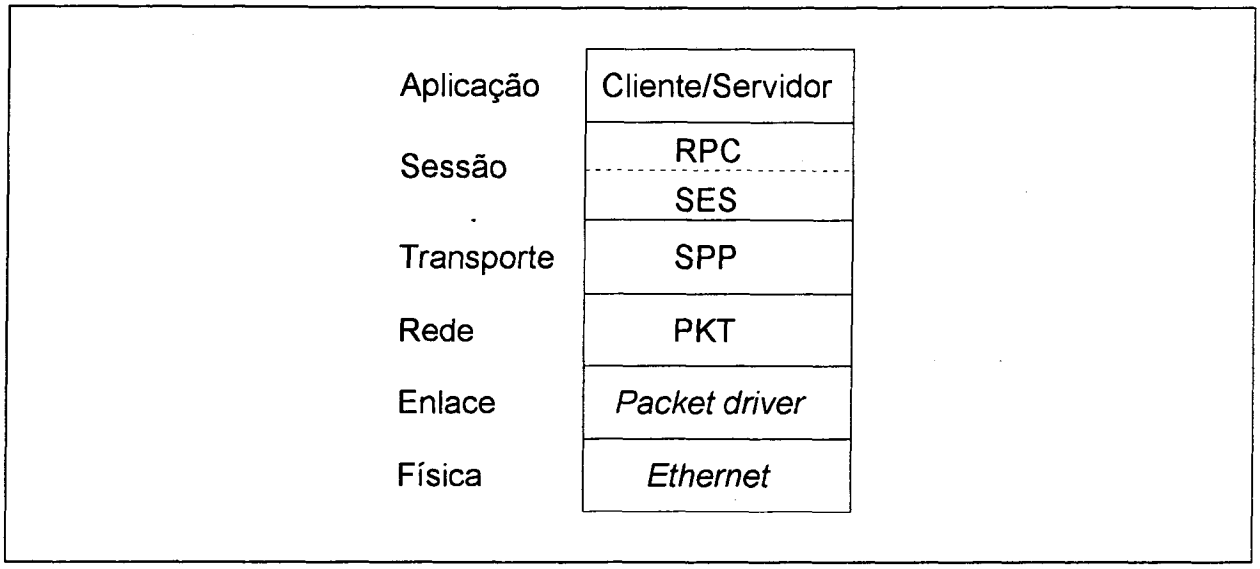

Figura 1.2- **Subsistema de Interconexão do Ambiente do LaSD.**

A primeira camada é o meio físico, já comentado no subitem Topologia, podendo maiores detalhes ser encontrado em Tanenbaum (1989).

A segunda camada é o packet driver, que é um protocolo de domínio público responsável pelo acesso ao meio físico. Este protocolo e sua documentação podem ser adquiridos através de FTP ANONYMOUS.

A camada seguinte é o protocolo PKT, o qual é responsável pelo emissão e recebimento dos pacotes e pela detecção de falha na comunicação ou em algum dos equipamentos através de um mecanismo de *time-out*.

Na próxima camada está o protocolo SPP que implementa a semântica no-máximo-uma-vez e é responsável pelo controle de processos/sessões.

O protocolo SES é responsável, se necessário, pela quebra da mensagem recebida do protocolo RPC (camada acima) em sub-mensagens com tamanhos limitados pelo tamanho máximo do pacote SPP e, também, pela passagem dessas sub-mensagens para o protocolo SPP.

O mecanismo de comunicação entre processos, alvo deste trabalho, é implementado na quinta camada. O mecanismo utilizado é a chamada de procedimento remoto ou RPC (*Remote Procedure Call*), escolhido pela sua facilidade de entendimento, pois tem a mesma filosofia da chamada de procedimento local.

Na última camada encontram-se as aplicações que utilizam os serviços do subsistema de interconexão. Nessa camada estão os processos clientes e servidores.

1.2- Chamada Remota de Procedimento

Existem vários mecanismos de comunicação entre processos na literatura e já implementados (como exemplo o send/receive simples, o redezvous, etc.), no entanto, o protocolo RPC é mais atrativo, pois possue características vantajosas sobre os demais, tais como: fácil entendimento, semelhança à chamada de procedimento local, e a familiaridade que os programadores já possuem com chamadas de procedimentos.

Funcionamento

Uma chamada de procedimento remoto é desempenhada da mesma forma que uma chamada local, ou seja, o processo que chama o procedimento pode passar parâmetros e é bloqueado até que o procedimento retorne com os resultados (figura 1.3). O processo que chama o procedimento é chamado de processo cliente e o processo que contém o procedimento chamado é chamado de processo servidor. A diferença principal entre a chamada remota e a chamada local é que na primeira o

procedimento chamado pode ser executado em uma máquina diferente do processo cliente.

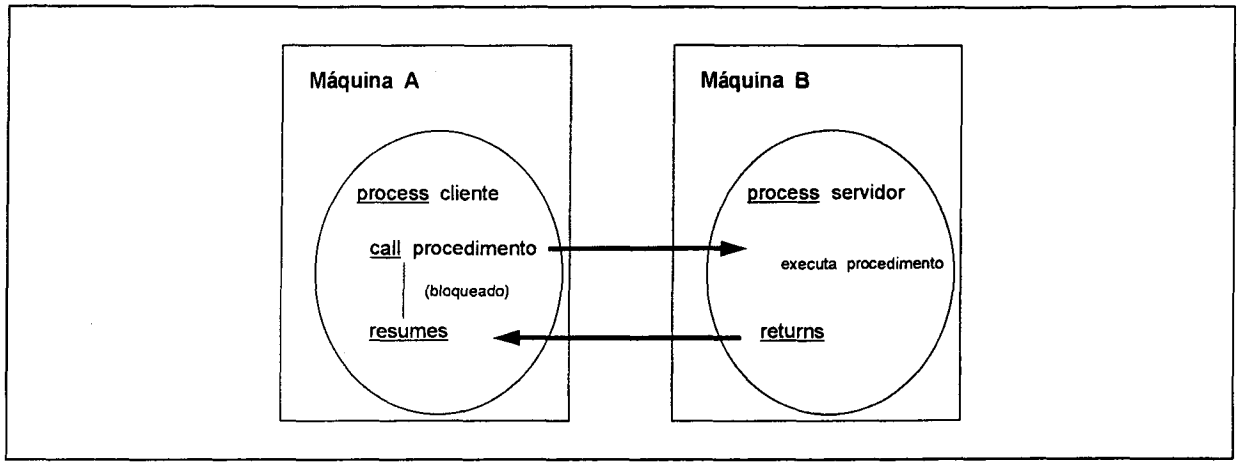

Figura 1.3- **Funcionamento do mecanismo RPC.**

Para que o processo cliente consiga executar o processo servidor em outra máquina, aquele precisa utilizar os serviços oferecidos pelo subsistema de comunicação. Ou seja, naquele, o processo cliente, em lugar de chamar diretamente o procedimento como nas chamadas locais, chama os serviços do subsistema de comunicação.

Características

Como pôde ser visto, as chamadas remotas são semelhantes às locais, porém existem algumas diferenças, com isto surgem características que devem ser levadas em consideração, tais como:

1. Semântica - forma pela qual o mecanismo RPC trata as perdas de mensagens e falhas na estação de trabalho ou no servidor. Como comentado anteriormente, o protocolo PKT possui um mecanismo de *time-out*, com o qual o

processo, ao enviar alguma mensagem, aguarda um tempo pré-determinado e, então, caso não receba confirmação de que o outro processo recebeu a mensagem, retransmite-a. Existem basicamente três semânticas: na primeira, chamada no-mínimo-uma-vez, o servidor sempre processa a requisição do cliente, mesmo que seja uma retransmissão, ou seja, o procedimento chamado é executado pelo menos uma vez; na segunda, chamada no-máximo-uma-vez, o servidor não processa a retransmissão, porém a semântica não garante que o procedimento solicitado será executado; e na terceira, chamada exatamente-uma-vez, o servidor sempre processa o procedimento e garante ser executado uma única vez;

2. *Binding* - o modo como o processo cliente descobre a máquina (ou melhor, o endereço da máquina) onde está o procedimento chamado. Essa descoberda pode ser feita de uma forma estática, na qual o processo cliente é compilado com o endereço da máquina que está o procedimento ou, de uma forma mais flexível, utilizando um arquivo ou uma tabela, no qual estão os procedimentos e suas localizações ou ainda, de maneira ideal, através de um *binder* (servidor de nomes);

3. Manipulação de parâmetros - visto que o procedimento remoto tem um espaço de endereçamento diferente do processo cliente, a passagem de parâmetro em chamadas remotas só são feitas por valor, jamais por referência;

4. Protocolos - são as regras impostas aos processos cliente e servidor na hora da comunicação. Existem basicamente três protocolos: o protocolo R (*Request Protocol*), no qual o processo cliente envia uma requisição e não espera resposta do servidor; o protocolo RR (Request/Response Protocol) em que o cliente ao enviar uma requisição aguarda a resposta do servidor; e o protocolo RRA (Request/Response/ Acknowledgement protocol) que é semelhante ao protocolo RR, porém o servidor, ao retornar a resposta, aguarda uma mensagem de confirmação de recebimento enviada pelo cliente; e

5. Concorrência - como o servidor está projetado para tratar as requisições feitas simultaneamente.

Essas características descritas acima refletem alguns dos problemas enfrentados no desenvolvimento de mecanismos de comunicação entre processos em aplicações distribuídas e, consequentemente, no mecanismo RPC, por isso serão abordadas mais adiante e também no desenvolvimento do sistema gerador de *stubs*.

Procedimentos *Stubs*

Como foi comprovado anteriormente, as diferenças entre chamadas remotas e chamadas locais existem, no entanto, uma solução para minimizar ou até eliminar essas diferenças é a utilização dos procedimentos *stubs* (ou simplesmente *stubs*). Os procedimentos *stubs* vêm tornar transparente essas diferenças entre a chamada remota e a local, ou seja, eles são encarregados de tratar as diferenças existentes.

Com a utilização dos *stubs*, os processos clientes, em lugar de solicitar serviços ao subsistema de comunicação, vão fazer chamadas aos procedimentos *stubs*, da mesma maneira que se fosse chamar localmente. A figura 1.4 mostra a estrutura de comunicação entre o processo cliente e o processo servidor utilizando os procedimentos *stubs*.

Observando, pode-se notar, então, que os problemas de comunicação entre processos em aplicações distribuídas, quando estruturado dessa maneira, passam a ser concentrados nos procedimentos *stubs*, ou seja, os stubs vão resolver as diferenças surgidas pela separação dos processos.

Com isso, a implementação dos *stubs* é o ponto crítico no desenvolvimento de aplicações distribuídas, visto que são rotinas que estão em contato direto com os protocolos do subsistema de comunicação.

Uma forma de facilitar a construção de aplicações distribuídas é utilizando um sistema gerador de *stubs*. O gerador de *stubs* é capaz de criar os procedimentos *stubs* a partir de uma especificação dos procedimentos que serão executados remotamente, assim, não é necessário se preocupar com protocolos de comunicação, fazendo com que se desenvolva aplicações distribuídas exatamente como se estivesse desenvolvendo aplicações em um sistema centralizado.

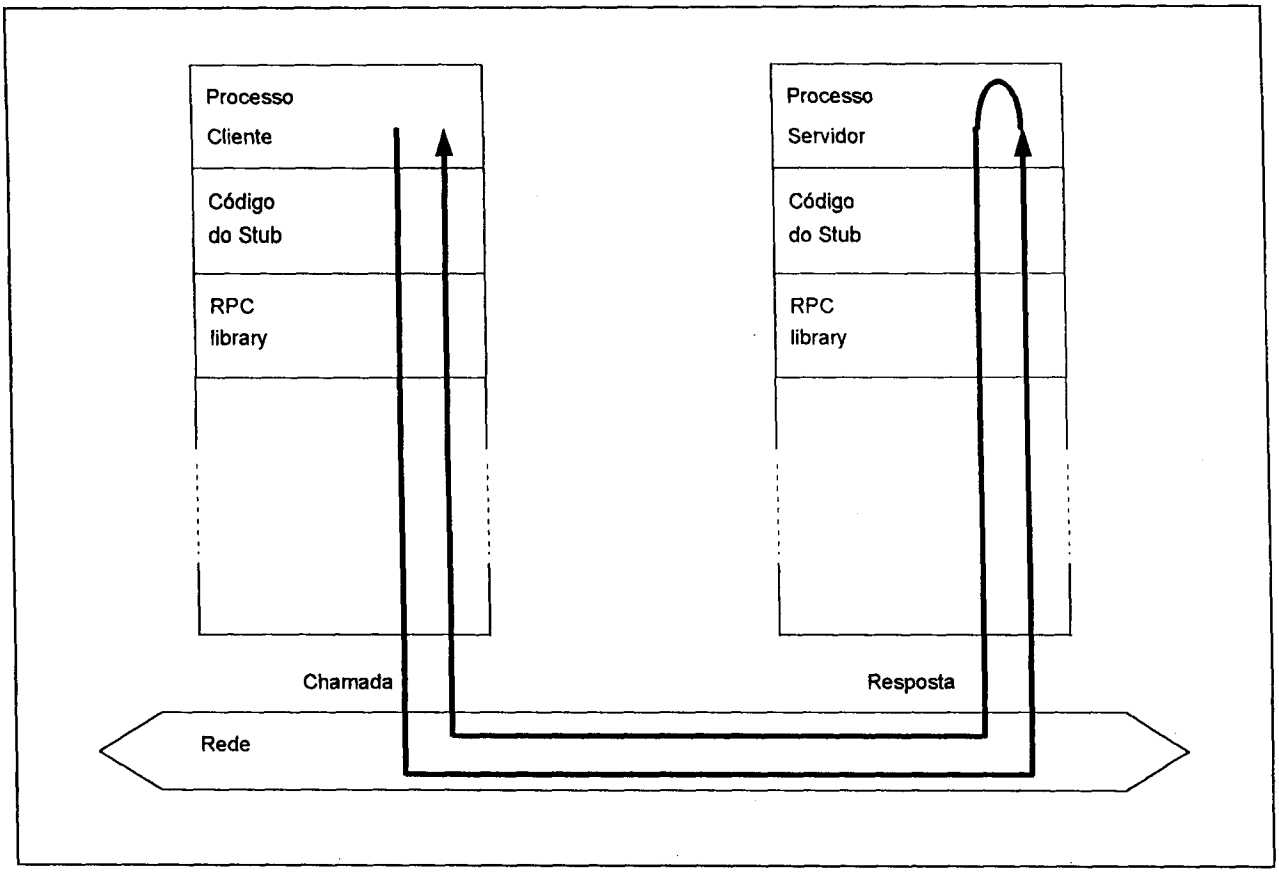

Figura 1.4- **RPC utilizando procedimentos stubs.**

Para facilitar ainda mais, deve-se desenvolver uma biblioteca RPC, cuja função é fazer a interface entre os procedimentos *stubs* e o subsistema de comunicação (como pode ser vista na figura 1.4). Com a biblioteca RPC, o procedimento *stub* passa a solicitar serviços às camadas abaixo através de rotinas implementadas para este propósito e que são colocadas em um único módulo.

A grande vantagem deste tipo de estruturação é a criação de rotinas padronizadas cada uma fazendo uma determinada função. Com isso, não há necessidade de se envolver com os protocolos das camadas abaixo quando for criar os *stubs*.

1.3- Considerações Finais

Nesta seção foram abordados pontos de extrema importância para dar prosseguimento ao projeto do gerador de *stubs*.

Foi mostrado o ambiente de *hardware* e de *software* em que será desenvolvido o sistema, as características do mecanismo RPC e o conceito e atuação do gerador de *stub*.

Na próxima seção será discutido o projeto lógico e físico do sistema GAPS.

SEÇÃO 2

ESPECIFICAÇÃO DO SISTEMA

Nesta seção encontra-se toda a especificação lógica e física do sistema gerador de stubs, o GAPS. A análise e o projeto foram baseados na análise estruturada de sistemas e no modelo relacional. Além da especificação funcional e de dados do sistema, encontra-se também uma visão orientada a objetos e a especificação da linguagem de entrada e do produto retornado pelo mesmo.

2.1- Propósito

O propósito do sistema é criar, a partir de uma linguagem de especificação, os procedimentos responsáveis pela comunicação entre o processo cliente e o processo servidor, utilizando os serviços do subsistema de comunicação.

2.2- Modelagem Funcional

O objetivo é o usuário definir os procedimentos (linguagem de interface), cujos *stubs* serão criados, e submeter ao sistema gerador. O diagrama da figura 2.1 está representando o objetivo do sistema.

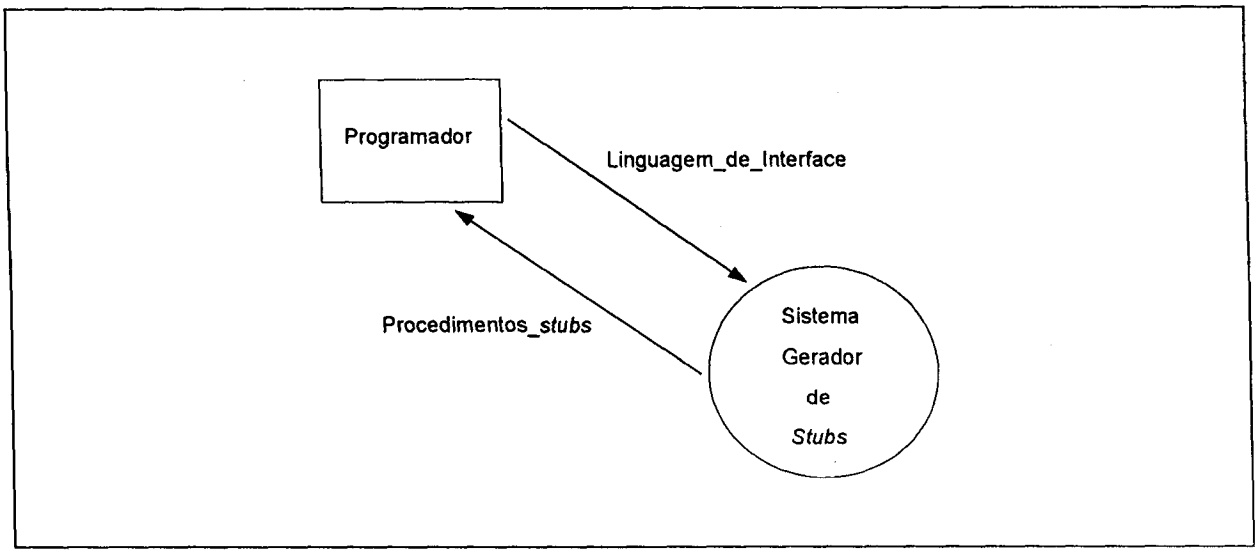

Figura 2.1- **Diagrama de Contexto.**

Mostrando a decomposição das funções do diagrama de contexto, encontram-se duas funções independentes: uma é a análise da linguagem de entrada e a outra é a criação dos procedimentos *stubs* (figura 2.2).

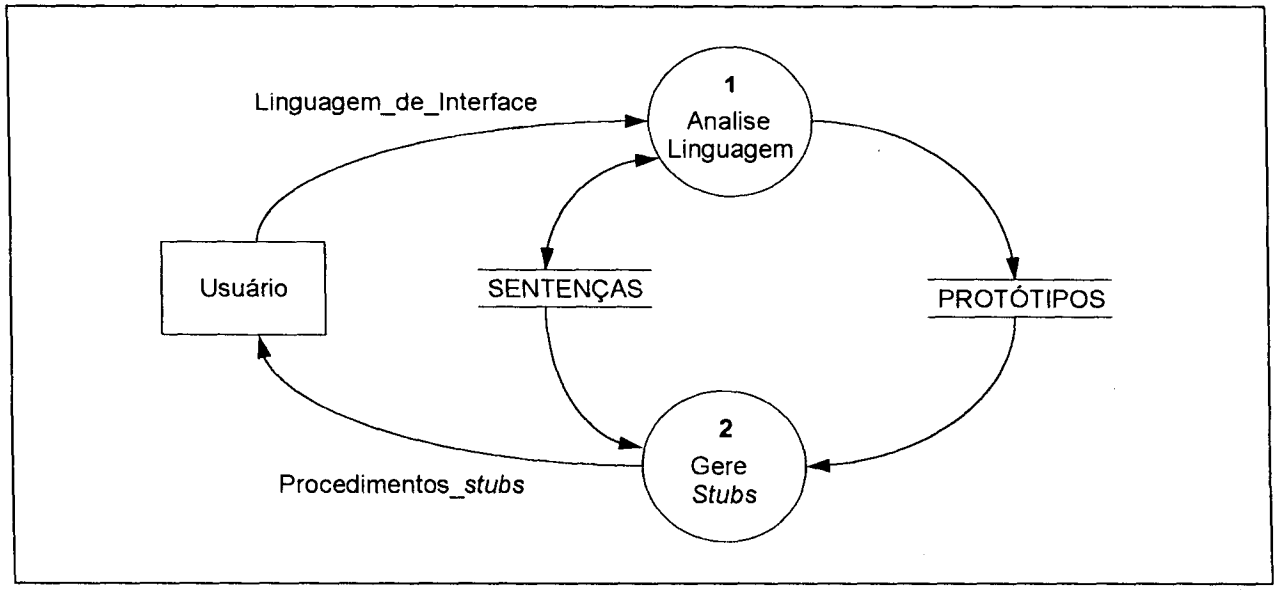

Figura 2.2- **Diagrama 0 (DFD - Diagrama de Fluxo de Dados).**

Explodindo a bolha 1 do diagrama 0, tem-se as seguintes funções:

1. A análise léxica, responsável pelo recebimento da linguagem de interface e classificação da linguagem em átomos;

2. A análise sintática, responsável pelo recebimento da lista de átomos, pela verificação das sintaxes e pela geração da lista de sentenças; e

3. A análise semântica, responsável pelo recebimento da lista de sentenças e pela verificação quanto ao contexto.

O diagrama pode ser visto na figura 2.3.

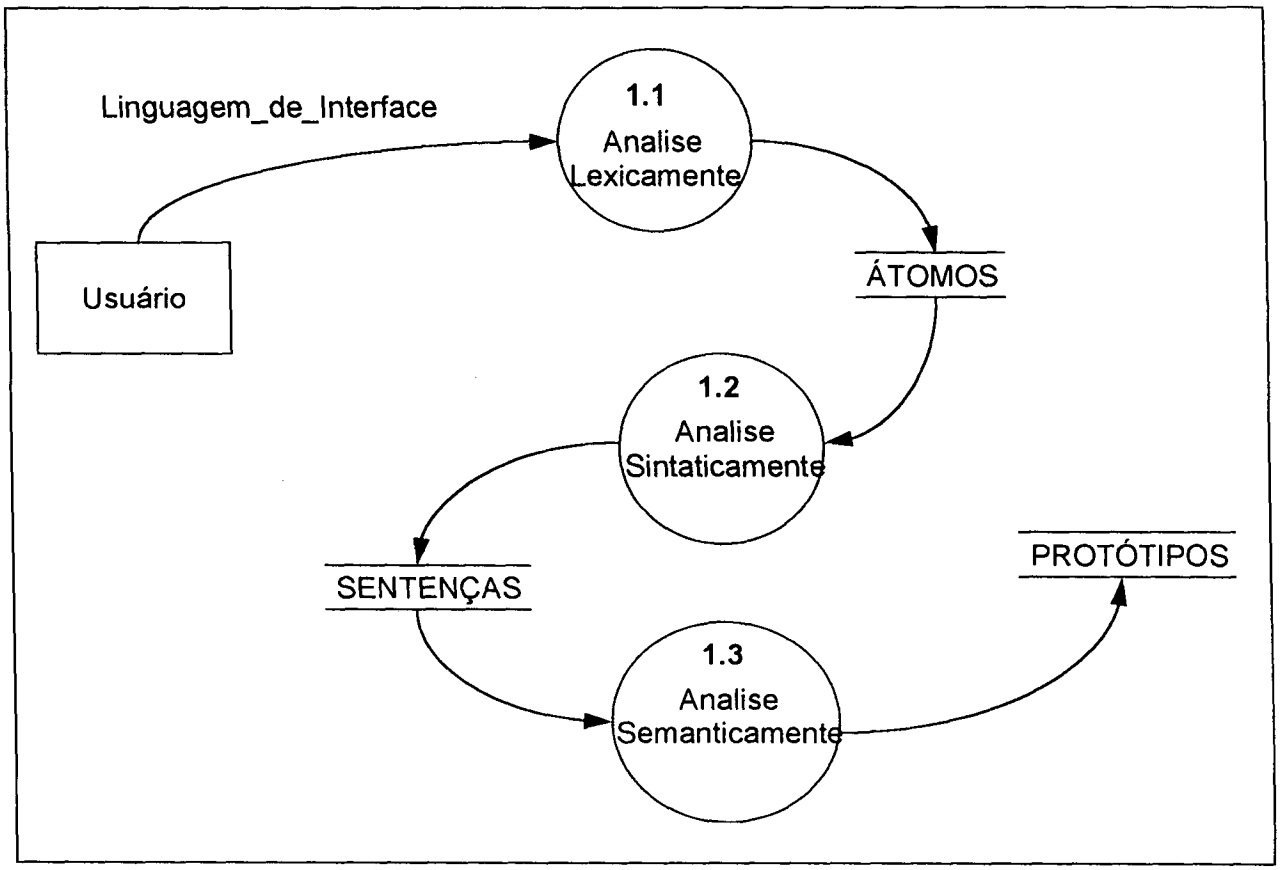

Figura 2.3- **Diagrama 1.**

A bolha 2 não se tem necessidade de explodir, visto que ela é responsável apenas pelo recebimento dos protótipos e pela criação dos procedimentos *stubs*.

A ferramenta utilizada foi baseada em Tom DeMarco (1989).

2.3- Modelagem de Dados

Na figura 2.4 constam as estruturas de infomormação do sistema e a associação entre elas.

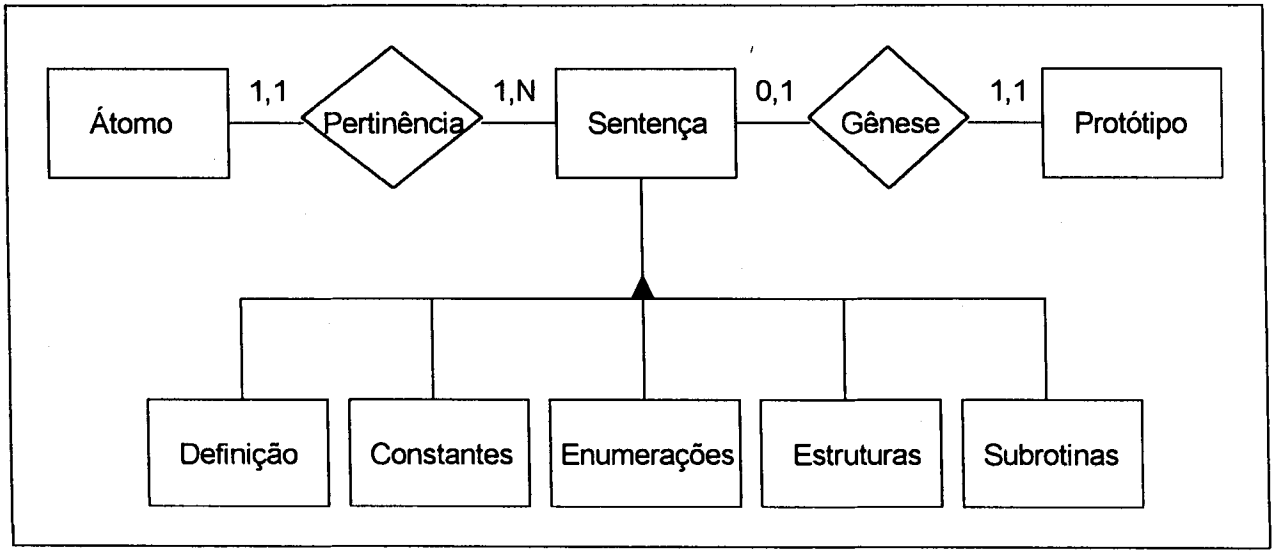

Figura 2.4- **Diagrama de Entidade e Relacionamento (DER).**

Entidade: **Átomo**

Atributos: Nome - nome do átomo
Classe - classe do átomo
NumOrdem - número de ordem do átomo
NumLinha - número da linha do átomo

Entidade: **Sentença**
Atributos: Nome - nome da sentença
Tipo - tipo da sentença
Entidade: **Protótipo**
Atributos: Nome - nome do protótipo
Tipo - tipo do protótipo

Relacionamentos:
Pertinência: um átomo pertence a uma única sentença,
uma sentença é composta por um ou mais átomos
Gênese: uma sentença gera um ou nenhum protótipo,
um protótipo é gerado por uma única sentença

Observação: Definição, Constante, Enumeração, Estrutura e Subrotina são especializações da entidade Sentença

A ferramenta utilizada foi baseada no diagrama de entidade e relacionamento, utilizado por Peter Chen.

2.4- Projeto Físico

A figura 2.5 mostra o diagrama de estrutura modular do sistema gerador de stubs. Como pode ser visto nesse modelo, são acrescentadas algumas características dependentes da implementação.

Este diagrama foi construído com base na metodologia utilizada por Page-Jones (1988).

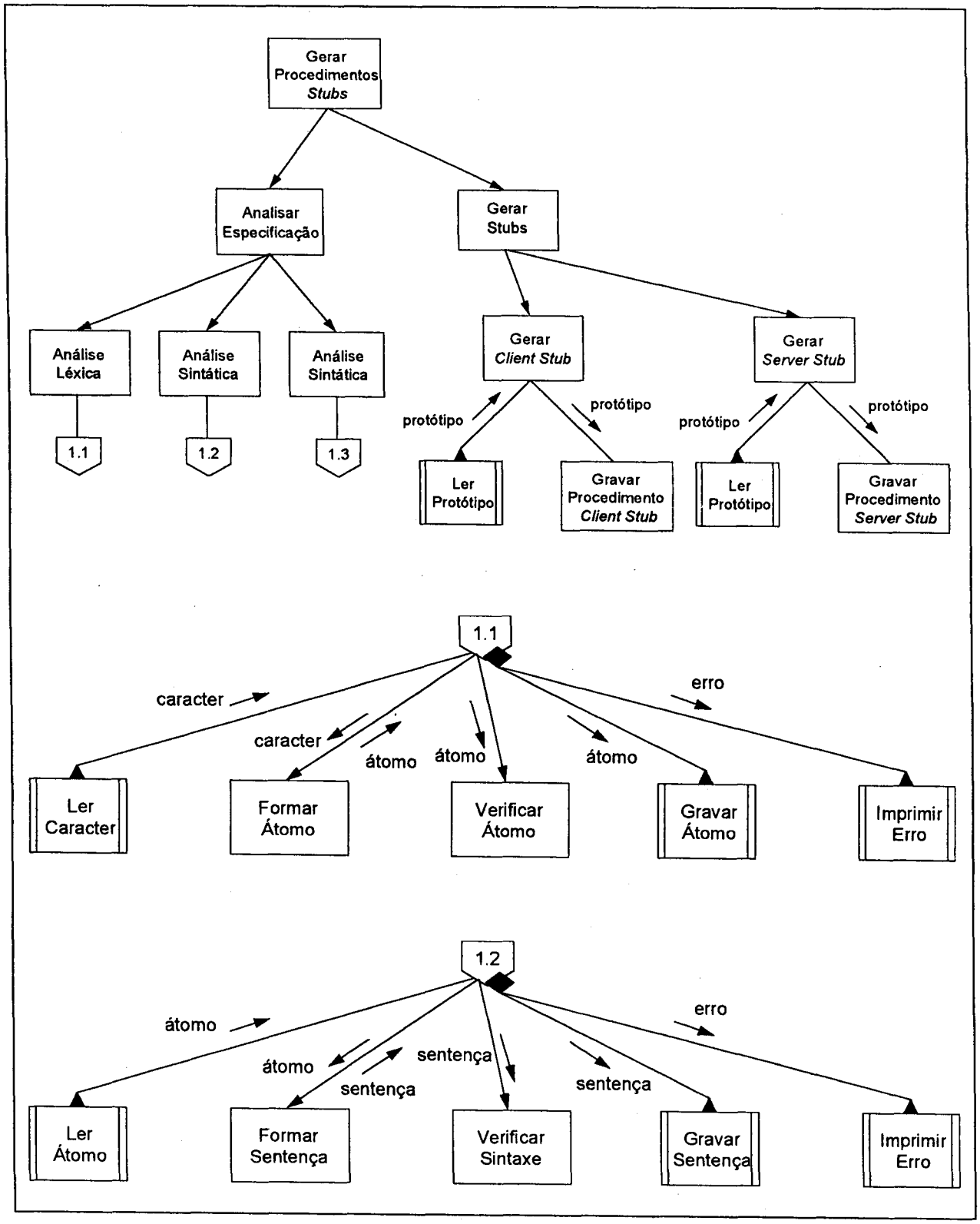

Figura 2.5- **Diagrama de Estrutura Modular (DEM).**

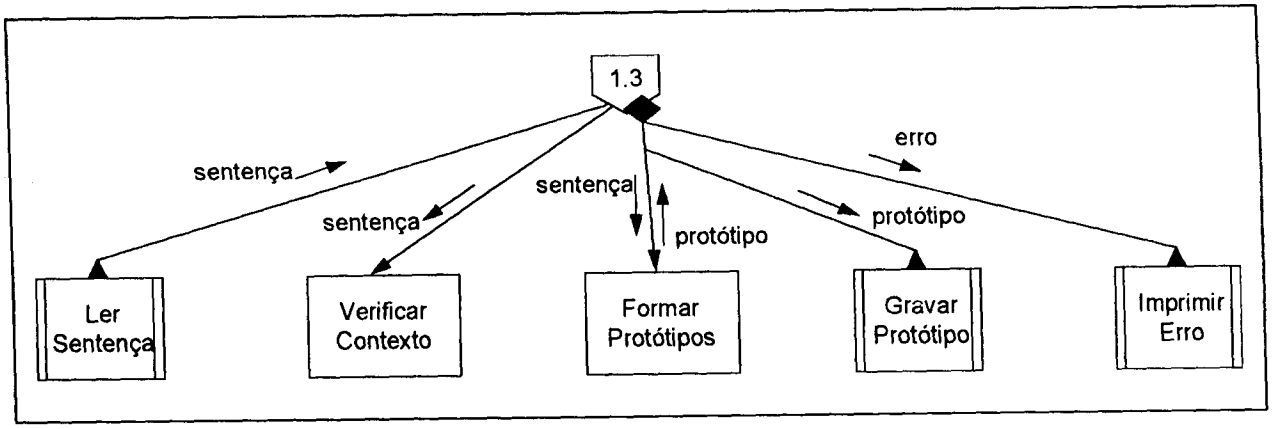

Figura 2.5- **Diagrama de Estrutura Modular (DEM).** (continuação)

2.5- Uma Visão Orientada a Objeto

Seguindo a tendência atual do mercado, com relação ao desenvolvimento de *softwares*, aprovitou-se este subitem para modelar o sistema gerador de *stubs* segundo a metodologia orientada a objeto.

Neste sentido foram criadas algumas classes e seus respectivos relacionamentos que estão descriminados abaixo e podem ser visualizados no diagrama da figura 2.6.

Classe: **Analisador** - objetos do tipo analisador
Atributos: ArquivoLGS - linguagem de interface
Métodos: Constroi - construtor
Destroi - destrutor
AnáliseLéxica - analisador léxico
FormaÁtomo - forma o átomo, caracter a caracter
AnáliseSintática - analisador sintático
FormaSentença - forma a sentença, átomo a átomo
AnáliseSemântica - analisador semântico
VerificaContexto - verifica átomos da sentença no contexto
FormaProtótipo - forma o protótipo da função ou procedimento

Relacionamentos: com Átomo

com Sentença

com Protótipo

Classe: **Gerador** - gerador dos módulos cliente e servidor

Atributos: ArquivoCLT - arquivo com os stubs do cliente

ArquivoSRV - arquivo com os stubs do servidor

Métodos: Constroi - construtor

Destroi - destrutor

Forma*Stub*CLT - forma os passos de cada procedimento cliente

Forma*Stub*SRV - forma os passos de cada procedimento servidor

Relacionamentos: com Sentença

com Protótipo

Classe: **Átomo** - objetos do tipo átomo

Atributos: Nome - nome do átomo

Classe - classe do átomo

NumOrdem - número de ordem do átomo

NumLinha - número da linha do átomo

Métodos: Constroi - construtor

Destroi - destrutor

AdicionarChar - adiciona caracter no átomo (FormaÁtomo)

Verifica - verifica validade do átomo

Classifica - classifica o átomo

Relacionamentos: com Especial

com Reservada

Classe: **Especial** - objetos do tipo caracter especial

Atributos: Nome - caracter especial

Métodos: Constroi - construtor

Destroi - destrutor

Éespecial - verifica se determinado átomo é caracter especial

Classe: **Reservada** - objetos do tipo palavra reservada

Atributos: Nome - palavra reservada

Métodos: Constroi - construtor

Destroi - destrutor

Éreservada - verifica se determinado átomo é palavra reservada

Classe: **Sentença** - objetos do tipo sentença (definição, constante, enumeração, estrutura e subrotinas)

Atributos: Nome - nome da sentença

Tipo - tipo da sentença

Métodos: Constroi - construtor

Destroi - destrutor

AdicionaÁtomo - adiciona átomo na sentença (FormaSentença)

Verifica - verifica validade da sentença

Tipo - retorna o tipo da sentença

Relacionamentos: com Átomo

Classe: **Protótipo** - objetos do tipo protótipos

Atributos: Nome - nome do protótipo

Tipo - tipo do protótipo

Métodos: Constroi - construtor

Destroi - destrutor

AdicionaÁtomo - adiciona átomo na sentença (FormaProtótipo)

Relacionamentos: com Átomo

com Argumento

Classe: **Argumento** - objetos do tipo argumento

Atributos: Tipo - tipo do argumento

Métodos: Constroi - construtor

Destroi - destrutor

Relacionamentos: com Átomo (Ser_Índice)

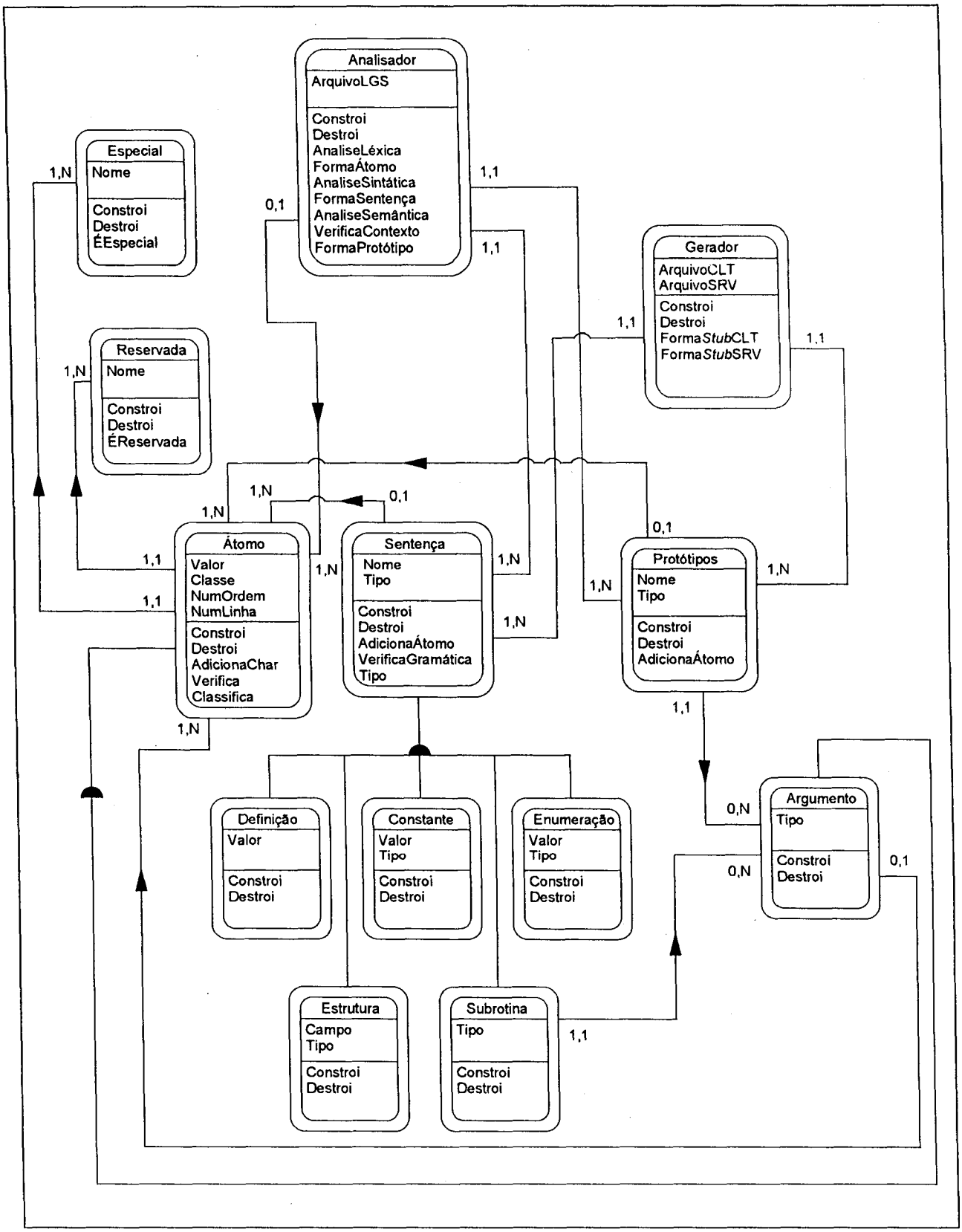

Figura 2.6- **Diagrama de Classes.**

2.6- Considerações Finais

Nesta seção foram descritas as funções e as estruturas que definem o sistema gerador de *stubs*.

Viu-se o objetivo do projeto e o que se precisa fazer para alcançá-lo; e, posteriormente, mostrou-se como atingir esse objetivo.

Finalizando a seção, o sistema foi modelado com base na metodologia orientada a objeto.

Na próxima seção serão vistas as especificações da linguagem de interface e dos procedimentos *stubs* gerados, além dos detalhes e restrições de implementação do GAPS.

SEÇÃO 3

IMPLEMENTAÇÃO DO SISTEMA

Esta seção é dedicada a detalhes de implementação do sistema GAPS e a restrições que lhe são impostas. No entanto, para ser um trabalho completo, precisa-se definir antes o fluxo de entrada do sistema (linguagem de interface) e o de saída (procedimentos *stubs*) e as rotinas de acesso ao subsistema de comunicação (biblioteca RPC).

3.1- Especificação da Linguagem de Interface

A linguagem de interface foi definida e formalizada através da Fórmula Normal de Backus (BNF) e está listada na figura 3.1.

Optou-se por uma linguagem com todas as característica possíveis do turbo-C, pois é a ferramenta de trabalho de todos os projetos do LaSD.

As diferenças são a forma de apresentação, algumas características adicionais que foram incorporadas e a algumas restrições que foram impostas à gramática da linguagem. Estas características podem ser verificadas na figura 3.2.

O primeiro ponto é a adição de uma estrutura, semelhante ao Pascal, que define o nome do módulo da linguagem de especificação, através da palavra reservada *definition*.

```
<LGS> := definition <identificador> { <bloco> }

<bloco> := [ <definição_de_constantes> ] |
           [ <definição_de_enumerações> ] |
           [ <definição_de_estruturas> ] |
             <definição_de_subrotinas>

<definição_de_constantes> := { const  <constante> = <número>; }*

<definicão_de_enumerações> := { enum <enumeração> { <const_enum> [ = <número> ]
                                  { , <const_enum> [ = <número> ] } }; }*

<definição_de_estruturas> := { struct <estrutura> { <tipo> <variável>;
                               { <tipo> <variável>; } }; }*


<definição_de_subrotinas> := { <definição_de_funções> [ : <servidor> ]; |
                               <definição_de_procedimentos> [ : <servidor> ]; }*

<definição_de_funções> :=  <tipo> <função> ( <argumentos> )

<definição_de_procedimentos> :=  [ void ] <procedimento> ( <argumentos> )

<argumentos> :=  void | <parâmetro> { , <parâmetro> }


<parâmetro> :=  <parâmetro_de_entrada> | <parâmetro_de_saída> |
                <parâmetro_de_entrada_saída>

<parâmetro_de_entrada> :=  [ in ] <tipo> <variável>

<parâmetro_de_saída> :=  out <tipo> <variável>

<parâmetro_de_entrada_saída> :=  inout <tipo> <variável>

<tipo> :=  <tipo_pre_definido> | string | enum <enumeração> | struct <estrutura>

<servidor> :=  <identificador>

<constante> :=  <identificador>

<enumeração> :=  <identificador>

<estrutura> :=  <identificador>
```

Figura 3.1- **Gramática da Linguagem de Interface.**

<procedimento> := <identificador>

<variável> := <identificador> | <identificador> [<número> | <identificador>]

<identificador> := (<letra> | <sublinhado>)
{ (<letra> | <sublinhado> | <dígito>) }

<letra> := **a** | **b** | **c** | ... | **z** | **A** | **B** | **C** | ... | **Z**

<sublinhado> := _

<número> := { <dígito> }*

<dígito> := **0** | **1** | **2** | ... | **9**

Figura 3.1- **Gramática da Linguagem de Interface.** (continuação)

Mais uma característica adicionada foi a incorporação da palavra reservada *string* utilizada para representar uma cadeia de caracter de tamanho máximo 255 e mais o \0 (fim de cadeia) totalizando 256 caracteres.

Na especificação dos argumentos foram incorporadas as palavras reservadas *in*, *out* e *inout*, as quais informam se o parâmetro é de entrada, de saída ou de entrada e saída da função ou procedimento, respectivamente. O *default* é *in*.

Com relação as restrições, a linguagem não aceita matrizes (*arrays* bidimensionais nem superiores), apenas vetores (*array* unidimensional), porém, na implementação do gerador, já existe estrutura para posterior utilização de matrizes.

Outra restrição é a impossibilidade do uso de ponteiros, visto que não existe passagem por referência em chamadas de procedimentos remotos, como foi comentado na sessão anterior.

Um ponto importante a ressaltar é a diferença entre uma *string* e uma cadeia de caracter. Uma *string* (mostrada no exemplo pela variável NOME) é limitada em 255

caracteres, no máximo, enquanto uma cadeia de caracteres (mostrada no exemplo pela variável ENDERECO) não tem limite.

```
/* linguagem de interface */
definition EXEMPLO
 {
  int SOMA(in double VET1[10], inout double VET2[10]);
  void CONSULTA(string NOME, out char ENDERECO[500]);
 }
```

Figura 3.2- **Exemplo da Linguagem de Interface.**

3.2- Especificação dos Procedimentos Stubs Gerados

A especificação do produto gerado é exatamente a gramática do turbo-C, sem adição nem restrição de características, pois são módulos que vão ser compilados e executados com a aplicação, utilizando o turbo-C. As figuras 3.3 e 3.4 mostram dois procedimentos *stub* gerados, um do cliente e outro do servidor.

No caso do *client stub*, o procedimento empacota os argumentos de entrada, chama o procedimento remoto (utilizando os recursos do subsistema de comunicação), recebe os resultados, desempacota-os e passa para o processo que chamou, tudo isto utilizando rotinas da biblioteca RPC.

No *server stub*, o procedimento recebe a requisição de execução do procedimento, desempacota os argumentos de entrada, chama o procedimento solicitado, recebe os resultados a serem retornados, empacota-os e envia-os ao processo que chamou, tudo isto também utilizando rotinas da biblioteca RPC.

```
int SOMA(double VET1[TAM],double VET2[TAM])
 { int TAM_MSG = 0;
   char *SRV = "1\0";
   BYTE MENSAGEM[3000];
   union TIPO_PRE ARGUMENTO;
   int I_1=0, J_1=0;
   int RETORNO;
   Inicializa();
   for (I_1=0; I_1<TAM; I_1++)
    { ARGUMENTO.DOUB = VET1[I_1];
      MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"double"); };
   for (I_1=0; I_1<TAM; I_1++)
    { ARGUMENTO.DOUB = VET2[I_1];
      MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"double"); };
   ExecutarProcedimento(SRV,10,PROC_01,MENSAGEM,&TAM_MSG);
   DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
   RETORNO = ARGUMENTO.INT;
   for (I_1=0; I_1<TAM; I_1++)
    { DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"double");
      VET2[I_1] = ARGUMENTO.DOUB; };
   return(RETORNO); }
```

Figura 3.3- **Exemplo do Módulo *Client Stub* da função SOMA.**

```
int Procedimento_02(int *TAM_MSGent, BYTE MENSAGEM[])
 { int TAM_MSG = 0;
   union TIPO_PRE ARGUMENTO;
   /* argumentos de entrada */
   char NOME[LEN_MAX_STR];
   char ENDERECO[500];
   int I_1=0, J_1=0;
   TAM_MSG = *TAM_MSGent;
   /* malloc */
   DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
   strcpy(NOME,ARGUMENTO.STRING);
   CONSULTA(NOME,ENDERECO);
   for (I_1=0; I_1<500; I_1++)
    { ARGUMENTO.CHAR = ENDERECO[I_1];
      MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char"); };
   *TAM_MSGent = TAM_MSG;
   /* free */
   return(0); }
```

Figura 3.4- **Exemplo do Módulo *Server Stub* da função CONSULTA.**

3.3- Definição da Biblioteca RPC

A última parte a ser definida para, então, implementar os gerador é criação da biblioteca RPC, a qual servirá de interface entre os procedimentos *stubs* e o subsitema de comunicação.

De acordo com a necessidade, foram construidas as seguintes rotinas:

1- MontarMensagem: coloca os parâmetros no pacote.

2- DesmontarMensagem: retira os parametros do pacote.

3- CarregaServidores: carrega a base de dados com os possíveis servidores da rede.

4- CarregaConfiguração: carrega o arquivo com os *handles* de cada conexão.

5- CarregaPrimitivas: carrega a estrutura de dados com as primitivas.

6- ObterServidor: retorna o endereço do servidor

7- GravaConfiguração: armazena os handles após a conexão.

8- Login: o cliente prepara o pacote com a solicitação de iniciar sessão no servidor.

9- Logout: o cliente prepara o pacote com a solicitação de finalizar sessão no servicor.

10- ExecutarProcedimento: responsável por obter informações necessárias para a transmissão do pacote e passa para o protocolo apropriado.

11- CriaSessão: inicia uma sessão no servidor para o cliente.

12- DestroiSessão: finaliza a sessão do cliente no servidor.

13- AnalisarPacote: verifica seqüencia, permissão, protocolo e sessão.

14- AtenderProcedimentoRemoto: recebe a requisição e executa a primitiva solicitada.

15- CarregaSessões: inicia a estrutura de dados responsável pelas informações sobre as sessões.

O implementação completa das rotinas da biblioteca RPC estão no apêndice A.

Foram criadas também algumas estruturas de dados que são manipuladas por algumas rotinas. São elas:

1- Tabela de Servidores: armazena os endereços dos possíveis servidores e seus respectivos protocolos e primitivas.

2- Tabela de Usuários: contém os nomes dos usuários, sua senha e um controle de permissão.

3- Tabela de Sessões: local responsável por gerenciar as sessões ativas no sistema.

3.4- Implementação do Sistema

Com a implementação da biblioteca RPC, pode-se partir para a declaração das estruturas utilizadas e, então, implementar o sistema gerador de *stubs*.

Estruturas Globais:

1- Tabela de Palavras Reservadas

2- Tabela de Caracteres Especiais

3- Tabela de Erros (léxicos, sintáticos e semânticos)

Estrutura do Analisador Léxico:

1- Lista de Átomos

```
struct TOKEN
  {
   struct TOKEN *ANTERIOR;   /* apontador para o átomo
                                anterior
                                */
```

```
    int ORDEM;                    /* número de ordem do
                                  átomo */
    char ATOMO[LEN_ATOMO];      /* nome do átomo */
    int CLASSE;                   /* classificação do átomo
                                  */
    int LINHA;                    /* número da linha em que
                                  o átomo
                                  se encontra */
      struct TOKEN *PROXIMO;  } /* apontador para o
                                  próximo átomo .
                                  */
  }

struct TOKEN  *LISTA_TOKEN = NULL;
```

Estruturas do Analisador Sintático:

1- Lista de Sentenças

Cada tipo de sentença tem uma estrutura semelhante.

```
struct <TIPO_SENTENCA>
  {
   <informação>;                  /* informações sobre a
                                     sentença */
   struct TOKEN *ENDERECO;        /* apontador para a
                                     lista de átomos */
   struct <TIPO_SENTENCA> *PROXIMO; /* apontador
                                       para a próxima
                                       sentença */
  }

struct <TIPO_SENTENCA> *<LISTA_TIPO_SENTENCA = NULL;
```

<TIPO_SENTENCA> - cada uma das sentenças (definição, constante, enumeração e estrutura) tem uma estrutura.

A subrotina é um caso especial e está declarada abaixo.

2- Lista de Subrotinas

```
struct INDICE /* lista de indices dos argumentos */
  {
   char IND[LEN_ATOMO];       /* nome ou valor do
                                 indice */
   struct INDICE *PROXIMO;    /* apontador para o
                                 próximo índice */
  }
```

```
struct ARGUMENTO /* lista de argumentos das
                    subrotinas */
 {
  int PARAMETRO;                /* in, out ou inout */
  char VARIÁVEL[LEN_ATOMO];     /* nome do argumento */
  char TIPO[LEN_ATOMO];         /* nome do tipo do
                                   argumento */
  int ENUM_ESTR;                /* indicador de
                                   enumeração ou
                                   estrutura */
  struct TOKEN *ENDERECO;       /* apontador para a
                                   lista de átomos */
  struct INDICE *PRIM_IND;      /* apontador para a
                                   lista de índices */
  struct ARGUMENTO *PROXIMO;    /* apontador para o
                                   próximo argumento */
 }

struct SUBROTINA
 {
  char NOME[LEN_ATOMO];         /* nome da subrotina */
  char TIPO[LEN_ÁTOMO];         /* nome do tipo da
                                   função */
  int ENUM_ESTR;                /* indicador de
                                   enumeração ou
                                   estrutura */
  char SERVIDOR[LEN_ATOMO];     /* nome do servidor
                                   que estará
                                   executando o
                                   procedimento */
  char PROTOCOLO[LEN_ATOMO];    /* tipo do protocolo
                                   de comunicação
                                   utilizado */
  struct TOKEN *ENDERECO;       /* apontador para a
                                   lista de átomos */
  struct ARGUMENTO *PRIM_ARG;   /* apontador para a
                                   lista de
                                   argumentos */
  struct SUBROTINA *PROXIMO;    /* apontador para a
                                   próxima
                                   subrotina */
 }

struct SUBROTINA *LISTA_SUBROTINA = NULL;
```

Estruturas do Analisador Semântico:

Utiliza as mesmas estruturas do analisador sintático.

Estrutura do Módulo Gerador dos Procedimentos *Stubs*:

1- Lista de Protótipos

```
struct PARAMETRO   /* lista de parâmetros */
  {
   int ENUM_ESTR;                /* indicador de
                                    enumeração ou
                                    estrutura */
   char NOME_TP[LEN_ATOMO];      /* nome do tipo enum
                                    ou struct */
   char TIPO_TP[LEN_ATOMO];      /* nome do tipo do
                                    parâmetro */
   char VARIAVEL[LEN_ATOMO]     /* nome do parâmetro */
   struct INDICE *PRIM_IND;      /* apontador para a
                                    lista de índices */
   struct ARGUMENTO *PROXIMO;    /* apontador para o
                                    próximo
                                    argumento */
  }

struct PROTOTIPOS
  {
   int NUM_PROT;
   struct PARAMETRO *RET;        /* apontador para a
                                    lista de
                                    parâmetros de
                                    retorno da
                                    função */
   struct PARAMETRO *ENT;        /* apontador para a
                                    lista de
                                    parâmetros de
                                    entrada */
   struct PARAMETRO *SAI;        /* apontador para a
                                    lista de
                                    parâmetros de
                                    saída */
   struct PARAMETRO *ENT_SAI;    /* apontador para a
                                    lista de
                                    parâmetros de
                                    entrada e saída */
   struct PROTOTIPO *PROXIMO;    /* apontador para o
                                    próximo
                                    protótipo */
  }

struct PROTOTIPOS *LISTA_PROTOTIPOS = NULL;
```

Pseudo-código do Analisador Léxico:

```
- Enquanto não fim-de-arquivo faça
     leia caracter
     se inicio-comentário
      então
        despreze texto até fim-de-comentário
      senão
        forme átomo
        se fim-de-átomo
          então
            se átomo for palavra reservada ou caracter
             especial ou
                        identificador ou número
              classifique átomo
              coloque na lista de átomos
             senão
              imprima erro léxico
            fim-se
         fim-se
      fim-se
  fim-enquanto
```

Pseudo-código do Analisador Sintático:

```
- Enquanto não fim-de-átomo faça
     pegue átomo
     se definition
      então
        caso
           a) átomo = define
              forme sentença-define
              se sentença = sintaxe da gramática
               então
                 coloque na lista de definições
               senão
                 imprima erro sintático
              fim-se
           b) átomo = const
              forme sentença-onst
              se sentença = sintaxe da gramática
               então
                 coloque na lista de constantes
               senão
                 imprima erro sintático
              fim-se
           c) átomo = enum
              forme sentença-enum
              se sentença = sintaxe da gramática
```

```
                então
                  coloque na lista de enumerações
                senão
                  imprima erro sintático
                fim-se
            d) átomo = struct
               forme sentença-struct
               se sentença = sintaxe da gramática
                então
                  coloque na lista de estruturas
                senão
                  imprima erro sintático
               fim-se
            e) átomo = procedimento ou função
               forme sentença-subrotina
               se sentença = sintaxe da gramática
                então
                  coloque na lista de subrotinas
                senão
                  imprima erro sintático
               fim-se
          fim-caso
        senão
          imprima erro de estruturação do módulo
      fim-se
    fim-enquanto
```

Pseudo-código do Analisador Semântico:

```
- Enquanto não fim-de-sentenças faça
    pega sentença
    Enquanto não fim-de-atomo
       pega átomo
       se precisa definição anterior
        então
          se não tem definição anterior
           então
             imprima erro semântico
          fim-se
        senão
          se já foi definido anteriormente
           então
             imprima erro semântico
          fim-se
       fim-se
    fim-enquanto
  fim-enquanto
```

Pseudo-código do Gerador de Código:

```
- Enquanto não fim-de-sentenças faça
    pega sentença
        se definição ou constante ou enumeração ou
estrutura
     então
       imprima sentença
    fim-se
  fim-enquanto
  Enquanto não fim-de-protótipo
    pega protótipo
    imprima client-stub
    imprima server-stub
  fim-enquanto
```

Os códigos implementados a partir destes pseudo-códigos estão no apêndice B.

3.5- Considerações Finais

Esta seção iniciou com a definição das linguagens de entrada (linguagem de interface) e de saída (procedimentos stubs). Posteriormente, foram implementadas as rotinas da biblioteca RPC e concluiu com a declaração das estruturas de dados envolvidas e com especificação e implementação de cada função do sistema.

A próxima seção contém algumas conclusões do desenvolvimento do sistema GAPS e também propostas para trabalhos futuros.

SEÇÃO 4

CONCLUSÃO

Esta seção finaliza o trabalho apresentando algumas conclusões importantes e sugestões para novos trabalhos.

4.1- Conclusão

Ficou evidente a importância deste trabalho, pois a literatura apenas comenta a existência de geradores de *stubs* e algumas características incorporadas pelos mesmos. Entretanto, não aborda detalhes do desenvolvimento, ou seja, não responde a: "como fazer um gerador de *stubs*?". Apesar de não ser um trabalho totalmente inédito, os pontos importantes relacionados à forma como os geradores são desenvolvidos o é, já que não foi encontrado nenhum trabalho a esse respeito. Por outro lado, os geradores encontrados são executados em outras plataformas diferentes da arquitetura da rede do LaSD. Este trabalho, veio, então, enriquecer o tema, trazendo mais conhecimento e motivando novas áreas de atuação.

O gerador de *stubs* é uma ferramenta de fundamental importância para quem deseja um aumento de produtividade no desenvolvimento de aplicações distribuídas. Isto acontece porque tira do programador a responsabilidade de criar sistemas que mexam diretamente com funções de redes, porque gerar *softwares* automaticamente é mais rápido do que criá-los manualmente e, também, porque os *softwares* gerados são praticamente livres de erros.

4.2- Sugestões para Trabalhos Futuros

Como sugestão de continuidade deste trabalho, segue abaixo algumas opções:

1- migrar este trabalho para outras plataformas;

2- migrar para o windows, aplicando a modelagem orientada a objeto, tanto no desenvolvimento, quanto na especificação do produto gerado;

3- projetar um gerador para ambientes heterogêneos;

4- incluir a utilização de matrizes;

5- estudar a possibilidade de aceitação, por parte do gerador, de ponteiros, através de alguma conversão, pois o mesmo só pode ser passado por valor; e

6- estudar a capacidade de memória de dados (área de globais e estáticas, área de pilha e área de heap) e a alocação de dados feita pelos procedimentos *stubs* e,posteriormente, projetar o gerador para que forneça procedimentos com a melhor utilização de memória possível.

BIBLIOGRAFIA

Aho, Alfred V., Sethi, Ravi e Ullman, Jeffrey D., **"Compilers - Principles, Techniques and Tools"**, *Addison-Wesley Publishing Company*, 1986.

Aho, Alfred V. e Ullman, Jeffrey D., **"Principles of Compilers Design"**, *Addison-Wesley Publishing Company*, 1977.

Bal, H. E.; Steiner, J. G. e Tanenbaum, A. S., **"Programming Languages for Distributed Computing Systems"**, *ACM Computing Surveys*, vol. 21, no 3, pág. 261-316, setembro, 1989.

Birrel, A. D. e Nelson, B. J., **"Implementing Remote Procedures Calls"**, *ACM Transactions on Computer Systems*, vol. 2, no 1, pág. 39-59, fevereiro, 1984.

Clarkson University, **"The Packet Driver Specification"**, User Documentation for the Packet Driver Collection, version 1.09, U.S., 1989.

Cleaveland, J. C., **"Building Application Generators"**, *IEEE Software*, vol. 5, no 4, pág. 25-33, julho, 1988.

Coulouris, G. F. e Dollimore, J., **"Distributed Systems: Concepts and Design"**, *Addison-Wesley Publishing Company*, 1988.

DeMarco, Tom, **"Análise Estruturada e Especificação de Sistema"**, *Editora Campus*, 2a edição, 1989.

Gane, Chris e Sarson, Trish, **"Análise Estruturada de Sistemas"**, *Editora LTC*, 8a edição, 1983.

José Neto, João, **"Introdução à Compilação"**, *Editora LTC*, 1987.

Kowaltowski, Tomasz, **"Implementação de Linguagens de Programação"**, *Editora Guanabara Dois*, 1983.

Lages, N. A. C. e Nogueira, J. M. S., **"Introdução aos Sistemas Distribuídos"**, *papirus Editora*, 1986.

Meira, C. A. A., **"Sobre Geradores de Aplicações"**, *Tese de Mestrado*, Universidade de São Paulo, ICMSC, 1991.

Page-Jones, Meilir, **"Projeto Estruturado de Sistemas"**, *McGraw-Hill*, 1988.

Setzer, Valdemar W., **"A Construção de um Compilador"**, *II Escola de Computação*, Campinas, 1981.

Shirley, J., **"Guide to Writing DCE Applications"**, *O'Reilly & Associates*, junho, 1992.

Stankelly-Bootle, **"Dominando o Turbo C"**, *Ed. Ciência Moderna*, 2a edição, 1989.

Sun Microsystems, **"Introduction to Remote Procedure Calls"**, *Network Programming*, pág 33-102 e 147-167, março, 1990.

Tanenbaum, A. S., **"Computer Networks"**, *Prentice-Hall*, 1989.

Tanenbaum, A. S. e van Renesse, R., **"Distributed Operating Systems"**, *ACM Computing Surveys*, vol. 17, no 4, pág. 419-470, 1985.

Tenenbaum, A. M, Langsam, Y. e Augenstein, M. **"Data Structures using C"**, *Prentice-Hall*, 1990.

Turbo C, **"User's Guide"**, *Borland International, Inc.*, versão 2.0.

Turbo C, **"Programmer's Guide"**, *Borland International, Inc.*, versão 2.0.

Veloso, P., Santos, C. dos, Azeredo, P. e Furtado, A., **"Estruturas de Dados"**, *Ed. Campus*, 4a edição, 1986.

Wilbur, S. e Bacarisse, B., **"Building distributed systems with remote procedure call"**, *IEE Software Engineering Journal*, vol. 2, no 5, pág. 148-159, setembro, 1987.

APÊNDICE A

Códigos de implementação das funções da Biblioteca RPC

Neste apêndice estão listadas as rotinas da biblioteca RPC do gerador de *stubs*, implementadas em turbo-C:

1- MontarMensagem: coloca os parâmetros no pacote.

```
void MontarMensagem(BYTE MENSAGEM[],int *TAMANHO,
                union TIPO_PRE *ARG,char *TIPO)
 {
  int CONTADOR1;
  int CONTADOR2;
  int TAM_STR = 0;
  int POSICAO;
  char ARGaux[DLenPacote] = "\0";
  BYTE *aux;
  CONTADOR1 = *TAMANHO;
  if (! strcmp(TIPO,"char"))
   {
    MENSAGEM[CONTADOR1++] = (BYTE) ARG->CHAR;
   }
  else
  if (! strcmp(TIPO,"unsigned char"))
   {
    MENSAGEM[CONTADOR1++] = (BYTE) ARG->UNSC;
   }
  else
  if (! strcmp(TIPO,"signed char"))
   {
    MENSAGEM[CONTADOR1++] = (BYTE) ARG->SIGC;
   }
  else
   {
    if (! strcmp(TIPO,"int"))
     {
      aux = (BYTE *) &ARG->INT;
      MENSAGEM[CONTADOR1++] = *aux;
```

```
    MENSAGEM[CONTADOR1++] = *(aux+1);
   }
  else
 if (! strcmp(TIPO,"unsigned int"))
  {
   aux = (BYTE *) &ARG->UNSI;
   MENSAGEM[CONTADOR1++] = *aux;
   MENSAGEM[CONTADOR1++] = *(aux+1);
  }
 else
if (! strcmp(TIPO,"signed int"))
 {
  aux = (BYTE *) &ARG->SIGI;
  MENSAGEM[CONTADOR1++] = *aux;
  MENSAGEM[CONTADOR1++] = *(aux+1);
 }
else
if (! strcmp(TIPO,"unsigned short int"))
 {
  aux = (BYTE *) &ARG->UNSSI;
  MENSAGEM[CONTADOR1++] = *aux;
 }
else
if (! strcmp(TIPO,"signed short int"))
 {
  aux = (BYTE *) &ARG->SIGSI;
  MENSAGEM[CONTADOR1++] = *aux;
 }
else
if (! strcmp(TIPO,"short int"))
 {
  aux = (BYTE *) &ARG->SINT;
  MENSAGEM[CONTADOR1++] = *aux;
 }
else
if (! strcmp(TIPO,"unsigned long int"))
 {
  aux = (BYTE *) &ARG->UNSLI;
  MENSAGEM[CONTADOR1++] = *aux;
  MENSAGEM[CONTADOR1++] = *(aux+1);
  MENSAGEM[CONTADOR1++] = *(aux+2);
  MENSAGEM[CONTADOR1++] = *(aux+3);
 }
else
if (! strcmp(TIPO,"signed long int"))
 {
  aux = (BYTE *) &ARG->SIGLI;
  MENSAGEM[CONTADOR1++] = *aux;
  MENSAGEM[CONTADOR1++] = *(aux+1);
  MENSAGEM[CONTADOR1++] = *(aux+2);
  MENSAGEM[CONTADOR1++] = *(aux+3);
 }
else
if (! strcmp(TIPO,"long int"))
 {
  aux = (BYTE *) &ARG->LINT;
```

```
  MENSAGEM[CONTADOR1++] = *aux;
  MENSAGEM[CONTADOR1++] = *(aux+1);
  MENSAGEM[CONTADOR1++] = *(aux+2);
  MENSAGEM[CONTADOR1++] = *(aux+3);
 }
else
 if (! strcmp(TIPO,"float"))
  {
   aux = (BYTE *) &ARG->FLOAT;
   MENSAGEM[CONTADOR1++] = *aux;
   MENSAGEM[CONTADOR1++] = *(aux+1);
   MENSAGEM[CONTADOR1++] = *(aux+2);
   MENSAGEM[CONTADOR1++] = *(aux+3);
  }
  else
   if (! strcmp(TIPO,"double"))
    {
     aux = (BYTE *) &ARG->DOUB;
     MENSAGEM[CONTADOR1++] = *aux;
     MENSAGEM[CONTADOR1++] = *(aux+1);
     MENSAGEM[CONTADOR1++] = *(aux+2);
     MENSAGEM[CONTADOR1++] = *(aux+3);
     MENSAGEM[CONTADOR1++] = *(aux+4);
     MENSAGEM[CONTADOR1++] = *(aux+5);
     MENSAGEM[CONTADOR1++] = *(aux+6);
     MENSAGEM[CONTADOR1++] = *(aux+7);
    }
    else
   if (! strcmp(TIPO,"long double"))
    {
     aux = (BYTE *) &ARG->LDOUB;
     MENSAGEM[CONTADOR1++] = *aux;
     MENSAGEM[CONTADOR1++] = *(aux+1);
     MENSAGEM[CONTADOR1++] = *(aux+2);
     MENSAGEM[CONTADOR1++] = *(aux+3);
     MENSAGEM[CONTADOR1++] = *(aux+4);
     MENSAGEM[CONTADOR1++] = *(aux+5);
     MENSAGEM[CONTADOR1++] = *(aux+6);
     MENSAGEM[CONTADOR1++] = *(aux+7);
     MENSAGEM[CONTADOR1++] = *(aux+8);
     MENSAGEM[CONTADOR1++] = *(aux+9);
     MENSAGEM[CONTADOR1++] = *(aux+10);
     MENSAGEM[CONTADOR1++] = *(aux+11);
     MENSAGEM[CONTADOR1++] = *(aux+12);
     MENSAGEM[CONTADOR1++] = *(aux+13);
     MENSAGEM[CONTADOR1++] = *(aux+14);
     MENSAGEM[CONTADOR1++] = *(aux+15);
    }
    else
     if (! strcmp(TIPO,"string"))
      {
       strcpy(ARGaux,ARG->STRING);
       POSICAO = CONTADOR1;
       CONTADOR1+=2;
       for (CONTADOR2 = 0; ARGaux[CONTADOR2] != '\0'; CONTADOR2++)
         MENSAGEM[CONTADOR1++] = (BYTE) ARGaux[TAM_STR++];
```

```
            /* empacota o tamanho da string */
            aux = (BYTE *) &TAM_STR;
            MENSAGEM[POSICAO++] = *aux;
            MENSAGEM[POSICAO] = *(aux+1);
           }
  }
 *TAMANHO = CONTADOR1;
}
```

2- DesmontarMensagem: retira os parametros do pacote.

```
void DesmontarMensagem(BYTE MENSAGEM[],int *TAMANHO,
                union TIPO_PRE *ARG,char *TIPO)
{
 int CONTADOR1 = 0;
 int CONTADOR2 = 0;
 int TAM_STR = 0;
 char ARGaux[DLenPacote] = "\0";
 BYTE *aux;
 if (! strcmp(TIPO,"char"))
   ARG->CHAR = (char) MENSAGEM[CONTADOR2++];
 else
 if (! strcmp(TIPO,"unsigned char"))
   ARG->UNSC = (unsigned char) MENSAGEM[CONTADOR2++];
 else
 if (! strcmp(TIPO,"signed char"))
   ARG->SIGC = (signed char) MENSAGEM[CONTADOR2++];
 else
   if (! strcmp(TIPO,"int"))
    {
     aux = (BYTE *) &ARG->INT;
     *aux = MENSAGEM[CONTADOR2++];
     *(aux+1) = MENSAGEM[CONTADOR2++];
    }
   else
   if (! strcmp(TIPO,"unsigned int"))
    {
     aux = (BYTE *) &ARG->UNSI;
     *aux = MENSAGEM[CONTADOR2++];
     *(aux+1) = MENSAGEM[CONTADOR2++];
    }
   else
   if (! strcmp(TIPO,"signed int"))
    {
     aux = (BYTE *) &ARG->SIGI;
     *aux = MENSAGEM[CONTADOR2++];
     *(aux+1) = MENSAGEM[CONTADOR2++];
    }
   else
   if (! strcmp(TIPO,"unsigned short int"))
    {
     aux = (BYTE *) &ARG->UNSSI;
     *aux = MENSAGEM[CONTADOR2++];
    }
```

```
 else
if (! strcmp(TIPO,"signed short int"))
 {
  aux = (BYTE *) &ARG->SIGSI;
  *aux = MENSAGEM[CONTADOR2++];
 }
 else
if (! strcmp(TIPO,"short int"))
 {
  aux = (BYTE *) &ARG->SINT;
  *aux = MENSAGEM[CONTADOR2++];
 }
 else
if (! strcmp(TIPO,"unsigned long int"))
 {
  aux = (BYTE *) &ARG->UNSLI;
  *aux = MENSAGEM[CONTADOR2++];
  *(aux+1) = MENSAGEM[CONTADOR2++];
  *(aux+2) = MENSAGEM[CONTADOR2++];
  *(aux+3) = MENSAGEM[CONTADOR2++];
 }
 else
if (! strcmp(TIPO,"signed long int"))
 {
  aux = (BYTE *) &ARG->SIGLI;
  *aux = MENSAGEM[CONTADOR2++];
  *(aux+1) = MENSAGEM[CONTADOR2++];
  *(aux+2) = MENSAGEM[CONTADOR2++];
  *(aux+3) = MENSAGEM[CONTADOR2++];
 }
 else
if (! strcmp(TIPO,"long int"))
 {
  aux = (BYTE *) &ARG->LINT;
  *aux = MENSAGEM[CONTADOR2++];
  *(aux+1) = MENSAGEM[CONTADOR2++];
  *(aux+2) = MENSAGEM[CONTADOR2++];
  *(aux+3) = MENSAGEM[CONTADOR2++];
 }
 else
  if (! strcmp(TIPO,"float"))
     {
      aux = (BYTE *) &ARG->FLOAT;
      *aux = MENSAGEM[CONTADOR2++];
      *(aux+1) = MENSAGEM[CONTADOR2++];
      *(aux+2) = MENSAGEM[CONTADOR2++];
      *(aux+3) = MENSAGEM[CONTADOR2++];
     }
   else
    if (! strcmp(TIPO,"double"))
     {
      aux = (BYTE *) &ARG->DOUB;
      *aux = MENSAGEM[CONTADOR2++];
      *(aux+1) = MENSAGEM[CONTADOR2++];
      *(aux+2) = MENSAGEM[CONTADOR2++];
      *(aux+3) = MENSAGEM[CONTADOR2++];
```

```
            *(aux+4) = MENSAGEM[CONTADOR2++];
            *(aux+5) = MENSAGEM[CONTADOR2++];
            *(aux+6) = MENSAGEM[CONTADOR2++];
            *(aux+7) = MENSAGEM[CONTADOR2++];
           }
          else
          if (! strcmp(TIPO,"long double"))
           {
            aux = (BYTE *) &ARG->LDOUB;
            *aux = MENSAGEM[CONTADOR2++];
            *(aux+1) = MENSAGEM[CONTADOR2++];
            *(aux+2) = MENSAGEM[CONTADOR2++];
            *(aux+3) = MENSAGEM[CONTADOR2++];
            *(aux+4) = MENSAGEM[CONTADOR2++];
            *(aux+5) = MENSAGEM[CONTADOR2++];
            *(aux+6) = MENSAGEM[CONTADOR2++];
            *(aux+7) = MENSAGEM[CONTADOR2++];
            *(aux+8) = MENSAGEM[CONTADOR2++];
            *(aux+9) = MENSAGEM[CONTADOR2++];
            *(aux+10) = MENSAGEM[CONTADOR2++];
            *(aux+11) = MENSAGEM[CONTADOR2++];
            *(aux+12) = MENSAGEM[CONTADOR2++];
            *(aux+13) = MENSAGEM[CONTADOR2++];
            *(aux+14) = MENSAGEM[CONTADOR2++];
            *(aux+15) = MENSAGEM[CONTADOR2++];
           }
          else
           if (! strcmp(TIPO,"string"))
            {
             /* desempacota o tamanho da string */
             aux = (BYTE *) &TAM_STR;
             *aux = MENSAGEM[CONTADOR2++];
             *(aux+1) = MENSAGEM[CONTADOR2++];
             for (CONTADOR1 = 0; CONTADOR1<TAM_STR; CONTADOR1++)
               ARGaux[CONTADOR1] = (char) MENSAGEM[CONTADOR2++];
             ARGaux[CONTADOR1] = '\0';
             strcpy(ARG->STRING,ARGaux);
            }
     for (CONTADOR1=0; CONTADOR2 != *TAMANHO; CONTADOR1++)
       MENSAGEM[CONTADOR1] = MENSAGEM[CONTADOR2++];
     *TAMANHO = CONTADOR1;
    }
```

3- CarregaServidores: carrega a base de dados com os possíveis servidores da rede.

```
void CarregaServidores(void)
 {
  FILE *ARQ_SRV;
  char PALAVRA[Str20] = "\0";
  char CAMINHO[Str20] = "\\gaps\\\0";
  char NOME_ARQ[Str40] = "\0";
  int I, J;
  strcpy(NOME_ARQ,CAMINHO);
```

```
 strcat(NOME_ARQ,"SERVIDOR.ES");
 if ((ARQ_SRV=fopen(NOME_ARQ,"r")) == NULL)
   ImprimeErro(406);
 /* ler os nomes e enderecos dos servidores */
 for (I = 0; I!=DMaxServ; I++)
  {
   /* ler o nome do servidor ate' encontrar um caracter branco */
   J = 0;
   for (;;)
    {
     PALAVRA[J] = fgetc(ARQ_SRV);
     if (PALAVRA[J] != '\n')
      {
       if (PALAVRA[J] == ' ')
        {
         PALAVRA[J] = '\0';
         break;
        }
       J++;
      }
    }
   for (J=0; PALAVRA[J]!='\0'; J++) PALAVRA[J] = toupper(PALAVRA[J]);
   /* coloca o nome do servidor na tabela de servidores */
   strcpy(TAB_SRV[I].NOME_SRV,PALAVRA);
   /* ler o endereco do servidor */
   for (J=0; J<DLenEnd; J++)
     fscanf(ARQ_SRV,"%2x",&(TAB_SRV[I].END_SRV[J]));
   /* iniciar sequencia e time-out para o servidor em cada primitiva */
   for (J=0; J<DMaxPrim; J++)
    {
     TAB_SRV[I].SEQUENCIA[J] = 0x00;
     TAB_SRV[I].TIME_OUT[J] = 65000;
    }
   /* iniciar sessao do servidor */
   TAB_SRV[I].SESSAO[0]= 0x00;
   TAB_SRV[I].SESSAO[1]= 0x00;
   /* iniciar handle do servidor */
   TAB_SRV[I].HANDLE = 0;
  }
 /* coloca o numero de servidores lidos em GNumServ */
 GNumServ = I;
 fclose(ARQ_SRV);
}
```

4- CarregaConfiguração: carrega o arquivo com os *handles* de cada conexão.

```
void CarregaConfiguracao(void)
 {
  FILE *ARQ_CFG;
  char LINHA[Str80] = "\0";
  char CAMINHO[Str20] = "\\gaps\\\0";
  char NOME_ARQ[Str40] = "\0";
  int I, J, SESSAOi;
  char SESSAO[2];
```

```
char HANDLE[5];
BYTE SEQ;
BYTE *aux;
strcpy(NOME_ARQ,CAMINHO);
strcat(NOME_ARQ,"ESTACAO.CFG");
if ((ARQ_CFG=fopen(NOME_ARQ,"r")) == NULL)
  ImprimeErro(3);
/* ler a sessao e o handle de cada servidor */
for (I = 0; I!=DMaxServ; I++)
 {
  /* ler a palavra 'servidor' e o nome do servidor */
  fgets(LINHA,80,ARQ_CFG);
  /* ler a palavra 'sessao' e a sessao */
  fgets(LINHA,80,ARQ_CFG);
  SESSAO[2] = '\0';
  SESSAO[0] = LINHA[7];
  SESSAO[1] = LINHA[8];
  SESSAOi = atoi(SESSAO);
  aux = (BYTE *) &SESSAOi;
  TAB_SRV[I].SESSAO[0] = *aux;
  SESSAO[0] = LINHA[9];
  SESSAO[1] = LINHA[10];
  SESSAOi = atoi(SESSAO);
  aux = (BYTE *) &SESSAOi;
  TAB_SRV[I].SESSAO[0] = *aux;
  /* ler a palavra 'handle' e o handle */
  fgets(LINHA,80,ARQ_CFG);
  HANDLE[4] = '\0';
  HANDLE[0] = LINHA[7];
  HANDLE[1] = LINHA[8];
  HANDLE[2] = LINHA[9];
  HANDLE[3] = LINHA[10];
  TAB_SRV[I].HANDLE = (int) atoi(HANDLE);
  /* ler a palavra 'sequencia' */
  fscanf(ARQ_CFG,"%10s",LINHA);
  /* ler as sequencias */
  for (J=0; J<200; J++)
    fscanf(ARQ_CFG,"%02x",&(TAB_SRV[I].SEQUENCIA[J]));
  LINHA[0] = fgetc(ARQ_CFG);
 }
fclose(ARQ_CFG);
/* imprime servidores e suas configuracoes */
printf("Servidores    Endereco          Sessao   Handle\n");
for (I=0; I<GNumServ; I++)
  {
   printf("%s",TAB_SRV[I].NOME_SRV);
   printf("       ");
   for (J=0; J<DLenEnd; J++)
     printf("%02x ",TAB_SRV[I].END_SRV[J]);
   printf("   %02x%02x",TAB_SRV[I].SESSAO[1],TAB_SRV[I].SESSAO[0]);
   printf("     %04d\n",TAB_SRV[I].HANDLE);
  }
}
```

5- CarregaPrimitivas: carrega a estrutura de dados com as primitivas.

```
void CarregaPrimitivas(void)
 {
  int I;
  /* carrega as primitivas de login e logout para todos os servidores */
  for (I = 0; I<GNumServ; I++)
   {
    TAB_PRI[I].NUM_PRIM = LOG_INN;
    TAB_PRI[I].NUM_PROT = SPP;
    strcpy(TAB_PRI[I].NOME_SRV,TAB_SRV[I].NOME_SRV);
    TAB_PRI[I+GNumServ].NUM_PRIM = LOG_OUT;
    TAB_PRI[I+GNumServ].NUM_PROT = SPP;
    strcpy(TAB_PRI[I+GNumServ].NOME_SRV,TAB_SRV[I].NOME_SRV);
   }
  GNumPrim = I + GNumServ;
  /* imprime tabela de primitivas */
  printf("Primitiva  Protocolo  Servidor GNumPri %d\n");
  for (I=0; I<GNumPrim; I++)
    {
     printf(" %3i       ",TAB_PRI[I].NUM_PRIM);
     printf("%4d      ",TAB_PRI[I].NUM_PROT);
     printf("%s\n",TAB_PRI[I].NOME_SRV);
    }
 }
```

6- ObterServidor: retorna o endereço do servidor

```
void ObterServidor(char *SRV, int *NUM_SRV, int PROT, int PRIM)
 {
  int I, J, EXISTE = 0;
  for (I=0; I<GNumPrim; I++)
   {
    if ((TAB_PRI[I].NUM_PRIM == PRIM) &&
       (TAB_PRI[I].NUM_PROT == PROT) &&
       (! strcmp(TAB_PRI[I].NOME_SRV,SRV)))
      {
       for (J=0; J<DMaxServ; J++)
         {
          if (! strcmp(TAB_SRV[J].NOME_SRV,SRV))
            {
             *NUM_SRV = J;
             EXISTE = 1;
             break;
            }
         }
      }
   }
  if (! EXISTE)
    ImprimeErro(407);
 }
```

7- GravaConfiguração: armazena os handles após a conexão.

```
void GravaConfiguracao(void)
 {
  FILE *ARQ_CFG;
  int I, J;
  char CAMINHO[Str20] = "\\gaps\\\0";
  char NOME_ARQ[Str40] = "\0";
  strcpy(NOME_ARQ,CAMINHO);
  strcat(NOME_ARQ,"ESTACAO.CFG");
  if ((ARQ_CFG=fopen(NOME_ARQ,"w")) == NULL)
    ImprimeErro(408);
  for (I=0; I<GNumServ; I++)
   {
    fprintf(ARQ_CFG,"servidor %s\n",TAB_SRV[I].NOME_SRV);
    fprintf(ARQ_CFG,"sessao ");
    fprintf(ARQ_CFG,"%02x%02x\n",TAB_SRV[I].SESSAO[1],
                    TAB_SRV[I].SESSAO[0]);
    fprintf(ARQ_CFG,"handle ");
    fprintf(ARQ_CFG,"%04i\n",TAB_SRV[I].HANDLE);
    fprintf(ARQ_CFG,"sequencia ");
    for (J=0; J<200; J++)
      fprintf(ARQ_CFG,"%02x",TAB_SRV[I].SEQUENCIA[J]);
    fprintf(ARQ_CFG,"\n");
   }
  fclose(ARQ_CFG);
 }
```

8- Login: o cliente prepara o pacote com a solicitação de iniciar sessão no servidor.

```
void Login(char *SRV, char *NOME, int SENHA)
 {
  int HANDLEaux, VERSAO, CLASSE, TIPO, NUMERO, BASICO,
     NUM_SRV, LEN_END, ERRO, NADA, K;
  BYTE END_EST[DLenEnd];          /* endereco da estacao */
  BYTE SESSAOaux[2];
  BYTE CADEIA[DLenPacote], *aux, ESTADO;
  int TAM_CADEIA, I, LENaux, J;
  CarregaServidores();
  CarregaConfiguracao();
  CarregaPrimitivas();
  ObterServidor(SRV,&NUM_SRV,SPP,LOG_INN);
  ERRO = ObterHandle(&HANDLEaux);
  if (ERRO)
   {
    /* termina o recebimento de pacotes do tipo ty */
    NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
    NADA++;
    ImprimeErro(ERRO);
   }
  /* coloca o novo handle na tabela de servidores */
  TAB_SRV[NUM_SRV].HANDLE = HANDLEaux;
```

```
/* obtencao do endereco Ethernet da estacao */
LEN_END = DLenEnd;
ERRO = ObterEndereco(GInt,HANDLEaux,END_EST,&LEN_END);
if (ERRO)
 {
  /* termina o recebimento de pacotes do tipo ty */
  NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
  NADA++;
  ImprimeErro(403);
 }
/* login no servidor */
/* zera sessao */
TAB_SRV[NUM_SRV].SESSAO[0] = 0x00;
TAB_SRV[NUM_SRV].SESSAO[1] = 0x00;
/* empacota conta do cliente */
strcpy(CADEIA,NOME);
for (I=strlen(NOME)+1; I<=Str20-2; I++)
  CADEIA[I] = ' ';
/* empacota senha do cliente */
aux = (BYTE *) &SENHA;
CADEIA[Str20-1] = *aux;
CADEIA[Str20] = *(aux+1);
/* empacota endereco da estacao */
CADEIA[Str20+1] = END_EST[0];
CADEIA[Str20+2] = END_EST[1];
CADEIA[Str20+3] = END_EST[2];
CADEIA[Str20+4] = END_EST[3];
CADEIA[Str20+5] = END_EST[4];
CADEIA[Str20+6] = END_EST[5];
/* empacota identificacao do processo */
aux = (BYTE *) &GIdeProc;
CADEIA[Str20+7] = *aux;
CADEIA[Str20+8] = *(aux+1);
TAM_CADEIA = 30;
ESTADO = TAB_SRV[NUM_SRV].SEQUENCIA[LOG_INN] & (BYTE) 0x80;
/* se primitiva de login ou de logout */
/* bit 6 = 1 (desprezar sequencia) */
ESTADO = ESTADO | (BYTE) 0x40;
SESSAOaux[0] = 0x00;
SESSAOaux[1] = 0x00;
ERRO = ChamarProcedimentoRemoto(TAB_SRV[NUM_SRV].END_SRV,
                  END_EST,GTipoPKT,
                  SPP,SESSAOaux,LOG_INN,ESTADO,
                  &TAM_CADEIA,CADEIA);
if (ERRO)
  {
   /* termina o recebimento de pacotes do tipo ty */
   NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
   NADA++;
   ImprimeErro(ERRO);
  }
/* complementar sequencia */
TAB_SRV[NUM_SRV].SEQUENCIA[LOG_INN] =
          (BYTE) (~TAB_SRV[NUM_SRV].SEQUENCIA[LOG_INN]);
SESSAOaux[0] = CADEIA[0];
SESSAOaux[1] = CADEIA[1];
```

```
 /* testa se codigo de erro e' diferente de zero */
 if (SESSAOaux[0])
  {
   /* termina o recebimento de pacotes do tipo ty */
   NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
   NADA++;
   ImprimeErro((int) SESSAOaux[0]);
  }
 /* coloca a nova sessao na tabela de servidores */
 TAB_SRV[NUM_SRV].SESSAO[0] = SESSAOaux[1];
 TAB_SRV[NUM_SRV].SESSAO[1] = SESSAOaux[0];
 /* termina o recebimento de pacotes do tipo ty */
 ERRO = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
 if (ERRO)
   ImprimeErro(404);
 for (J=0; J<DMaxPrim; J++)
   TAB_SRV[NUM_SRV].SEQUENCIA[J] = 0x00;
 GravaConfiguracao();
 printf ("\n\n    #####  Bom trabalho !!!  #####\n\n");
}
```

9- Logout: o cliente prepara o pacote com a solicitação de finalizar sessão no servicor.

```
/* interfaceamento com a rede para desconexao do servidor
   entrada: SRV -> nome do servidor */
void Logout(char *SRV)
 {
  int HANDLEaux, VERSAO, CLASSE, TIPO, NUMERO, BASICO,
     NUM_SRV, LEN_END, ERRO, NADA;
  BYTE END_EST[DLenEnd];          /* endereco da estacao */
  BYTE CADEIA[DLenPacote], ESTADO, SESSAOaux[2];
  int TAM_CADEIA, J;
  CarregaServidores();
  CarregaConfiguracao();
  CarregaPrimitivas();
  ObterServidor(SRV,&NUM_SRV,SPP,LOG_OUT);
  ERRO = ObterHandle(&HANDLEaux);
  if (ERRO)
   {
    /* termina o recebimento de pacotes do tipo ty */
    NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
    NADA++;
    ImprimeErro(ERRO);
   }
  /* coloca o novo handle na tabela de servidores */
  TAB_SRV[NUM_SRV].HANDLE = HANDLEaux;
  /* obtencao do endereco Ethernet da estacao */
  LEN_END = DLenEnd;
  ERRO = ObterEndereco(GInt,HANDLEaux,END_EST,&LEN_END);
  if (ERRO)
   {
    /* termina o recebimento de pacotes do tipo ty */
```

```
      NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
      NADA++;
      ImprimeErro(403);
     }
    /* logout no servidor */
    CADEIA[0] = TAB_SRV[NUM_SRV].SESSAO[0];
    CADEIA[1] = TAB_SRV[NUM_SRV].SESSAO[1];
    TAM_CADEIA = 2;
    ESTADO = TAB_SRV[NUM_SRV].SEQUENCIA[LOG_OUT] & (BYTE) 0x80;
    /* se primitiva de login ou de logout */
    /* bit 6 = 1 (desprezar sequencia) */
    ESTADO = ESTADO | (BYTE) 0x40;
    SESSAOaux[0] = TAB_SRV[NUM_SRV].SESSAO[0];
    SESSAOaux[1] = TAB_SRV[NUM_SRV].SESSAO[1];
    ERRO = ChamarProcedimentoRemoto(TAB_SRV[NUM_SRV].END_SRV,
                    END_EST,GTipoPKT,
                    SPP,SESSAOaux,LOG_OUT,ESTADO,
                    &TAM_CADEIA,CADEIA);
    if (ERRO)
      {
       /* termina o recebimento de pacotes do tipo ty */
       NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
       NADA++;
       ImprimeErro(ERRO);
      }
    /* complementar sequencia */
    TAB_SRV[NUM_SRV].SEQUENCIA[LOG_OUT] =
              (BYTE) (~TAB_SRV[NUM_SRV].SEQUENCIA[LOG_OUT]);
    /* testa se codigo de erro e' diferente de zero */
    if (CADEIA[0])
     {
      /* termina o recebimento de pacotes do tipo ty */
      NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
      NADA++;
      ImprimeErro(7);
     }
    /* zera a sessao na tabela de servidores */
/*  TAB_SRV[NUM_SRV].SESSAO[0] = CADEIA[0];
    TAB_SRV[NUM_SRV].SESSAO[1] = CADEIA[1]; */
    /* termina o recebimento de pacotes do tipo ty */
    ERRO = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
    if (ERRO)
      ImprimeErro(404);
    for (J=0; J<DMaxPrim; J++)
      TAB_SRV[NUM_SRV].SEQUENCIA[J] = 0x00;
    GravaConfiguracao();
    printf ("\n\n    #####  Fim de trabalho !!!  #####\n\n");
   }
```

10- ExecutarProcedimento: responsável por obter informações necessárias para a transmissão do pacote e passa para o protocolo apropriado.

```
int ChamarProcedimentoRemoto(BYTE END_SRV[],
```

```
                    BYTE END_CLT[],
                    BYTE TIPO_PKT[],
                    BYTE PROT, BYTE SESSAO[],
                    BYTE PRIM, BYTE ESTADO,
                    int *TAM_DADOS,
                    BYTE DADOS[])

 {
  BYTE *aux, ENDERaux[DLenEnd], ENDERaux1[DLenEnd];
  int INDICE, ERRO, NADA, TAMaux;
  if (PROT == SPP)
   {
    for (INDICE=0; INDICE<DLenEnd; INDICE++)
      ENDERaux[INDICE] = END_SRV[INDICE];
    for (INDICE=0; INDICE<DLenEnd; INDICE++)
      ENDERaux1[INDICE] = END_CLT[INDICE];
    ERRO = TransmitirSPP_toSRV(ENDERaux,ENDERaux1,TIPO_PKT,
                    &PROT,SESSAO,PRIM,ESTADO,
                    TAM_DADOS,DADOS);
    if (ERRO)
      return(ERRO);
    if (PROT != SPP)
      return(502);
   }
  else
    return(501);
 return(0);
 }


void ExecutarProcedimento(char *SRV, int PROT, int PRIM,
                BYTE ARGUMENTOS[], int *TAM_ARGUMENTOS)
 {
int I,J;
  int HANDLEaux, VERSAO, CLASSE, TIPO, NUMERO, BASICO,
    NUM_SRV, LEN_END, ERRO, NADA;
  static int TamSendBufaux; /* copia Tamanho do buffer a ser efetivamente transmitido */
  static BYTE END_EST[DLenEnd];          /* endereco da estacao */
  static BYTE ESTADO, SESSAOaux[2];
  static BYTE SendBuf[DLenPacote]; /* Tam. do Buffer de Transm */
  static int TamSendBuf; /* Tamanho do buffer a ser efetivamente transmitido */
  static BYTE SendBufaux[3000]; /* copia do Buffer de Transm */
  div_t NumPacotes;
  int INDICE, NumPacRec;
  BYTE *aux;
  BYTE EndSer[DLenEnd], EndClt[DLenEnd], TipoPKT[2];
  BYTE Protocolo='0',Processo[2], Primitiva = '0', Sergipe='0';

 /* imprime tabela de primitivas */
 printf("Indice  Primitiva  Protocolo  Servidor === teste\n");
 for (I=0; I<GNumPrim; I++)
 {
   printf(" %1i     ",I);
   printf(" %3i      ",TAB_PRI[I].NUM_PRIM);
   printf("%4d     ",TAB_PRI[I].NUM_PROT);
   printf("%s\n",TAB_PRI[I].NOME_SRV);
```

```
  }
/* imprime servidores e suas configuracoes */
printf("Indice  Servidores    Endereco          Sessao   Handle === teste\n");
for (I=0; I<GNumServ; I++)
  {
   printf("  %1i    ",I);
   printf("%s",TAB_SRV[I].NOME_SRV);
   printf("        ");
   for (J=0; J<DLenEnd; J++)
     printf("%02x ",TAB_SRV[I].END_SRV[J]);
   printf("    %02x%02x",TAB_SRV[I].SESSAO[1],TAB_SRV[I].SESSAO[0]);
   printf("     %04d\n",TAB_SRV[I].HANDLE);
  }

   printf ("\n\n    #####  Inicio de Conexao !!!  #####\n\n");

printf("Servidor: %s - Protocolo: %i - Primitiva: %i\n",
     SRV,PROT,PRIM);

   ObterServidor(SRV,&NUM_SRV,PROT,PRIM);

printf("Numero do servidor: %d\n",NUM_SRV);

   ERRO = ObterHandle(&HANDLEaux);
   if (ERRO)
    {
     /* termina o recebimento de pacotes do tipo ty */
     NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
     NADA++;
     ImprimeErro(ERRO);
    }
   /* coloca o novo handle na tabela de servidores */
   TAB_SRV[NUM_SRV].HANDLE = HANDLEaux;
   /* obtencao do endereco Ethernet da estacao */
   LEN_END = DLenEnd;
   ERRO = ObterEndereco(GInt,HANDLEaux,END_EST,&LEN_END);
   if (ERRO)
    {
     /* termina o recebimento de pacotes do tipo ty */
     NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
     NADA++;
     ImprimeErro(403);
    }

   /* 1490 = TAM. DO PKT - 24 */
   NumPacotes = div (*TAM_ARGUMENTOS, 1490);
   if ((NumPacotes.rem) || (*TAM_ARGUMENTOS == 0))  NumPacotes.quot++;
   aux = (BYTE *) &(NumPacotes.quot);
   SendBuf[0] = *aux;
   SendBuf[1] = *(aux+1);
   INDICE = 0;
   while ((INDICE < *TAM_ARGUMENTOS) || (INDICE==0)){
    ESTADO = TAB_SRV[NUM_SRV].SEQUENCIA[PRIM] & (BYTE) 0x80;
    SESSAOaux[0] = TAB_SRV[NUM_SRV].SESSAO[0];
    SESSAOaux[1] = TAB_SRV[NUM_SRV].SESSAO[1];
    TamSendBuf = 2;
```

```
    while ((TamSendBuf < 1490) && (INDICE < *TAM_ARGUMENTOS)){
      SendBuf[TamSendBuf] = ARGUMENTOS[INDICE];
      TamSendBuf++;
      INDICE++;
     } /* Fim while 1 */
    if ((INDICE == 0) && (*TAM_ARGUMENTOS == 0)) INDICE++;
    TamSendBufaux = TamSendBuf;
    for (I=0; I<TamSendBuf; I++)
       SendBufaux[I] = SendBuf[I];
    ERRO = ChamarProcedimentoRemoto(TAB_SRV[NUM_SRV].END_SRV,
                          END_EST,GTipoPKT,
                          PROT,
                          SESSAOaux,
                          PRIM,ESTADO,
              &TamSendBufaux, SendBufaux);
    if (ERRO) {
      /* termina o recebimento de pacotes do tipo ty */
      NADA = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
      NADA++;
      ImprimeErro(ERRO);
     }
    /* complementar sequencia */
    TAB_SRV[NUM_SRV].SEQUENCIA[PRIM] =
                             (~TAB_SRV[NUM_SRV].SEQUENCIA[PRIM]);

  } /* FIM WHILE 0 */


  /* TamSendBufaux, SendBufaux: Tamanho do prim. buffer de resposta */
  /*   e valore que deve ser passados para a primitiva            */
  aux = (BYTE*) &NumPacRec;
  *aux = SendBufaux[0];
  *(aux+1) = SendBufaux[1];
  for (J=0; J<TamSendBufaux-2; J++)
    ARGUMENTOS[J] = SendBufaux[J+2];
  INDICE = J;
  NumPacRec--;
  while  (NumPacRec){
    ReceberSPP_fromSRV(EndSer, EndClt, TipoPKT, &Protocolo, Processo,
                              &Primitiva, &Sergipe, &TamSendBufaux, SendBufaux);
    for (J=0; J<TamSendBufaux-2; J++)
         ARGUMENTOS[INDICE+J] = SendBufaux[J+2];
    INDICE += J;
    NumPacRec--;
  };
  *TAM_ARGUMENTOS = INDICE;

  /*  Recebeu todos os pacotes => finalizar primitiva */
  ERRO = TerminarAcessoTipoPacote(GInt,TAB_SRV[NUM_SRV].HANDLE);
  if (ERRO)
    ImprimeErro(404);

  GravaConfiguracao();
  printf ("\n\n    #####  Fim de Conexao !!!  #####\n\n");
}
```

11- CriaSessão: inicia uma sessão no servidor para o cliente.

```
int CriaSessao(int *TAM_BUFFER, BYTE *BUFFER)
  {
   BYTE *aux, SESaux[2];
   int I, J, K, USUaux;
   char NOME[Str20];
   printf("\n*** inicio do login ***\n");
   SESaux[1] = 0;
   SESaux[0] = 0;
   strncpy(&(NOME[0]),&(BUFFER[0]),20);
   NOME[Str20-1] = '\0';
   printf("nome: %s\n", NOME);
   /* verifica se o usuario existe */
   for (J=1; J<=DMaxUsu; J++)
     if (! strcmp(TAB_USU[J].NOME_USU,NOME))
       break;
   if (J > DMaxUsu)
     /* usuario nao existe */
     SESaux[0] = 2;
    else
     {
      /* usuario existe */
      USUaux = J;
      /* verifica se a estacao ja esta conectada */
      for (I=1; I<=DMaxSes; I++)
        /* verifica se ha usuario na sessao */
        if (TAB_SES[I].COD_USU != 0)
         {
          for (K=0; K<DLenEnd; K++)
            /* verifica se o endereco da estacao do usuario que esta'
               tentando se conectar e' o mesmo do ja' esta' conectado */
            if (TAB_SES[I].END_EST[K] != BUFFER[Str20+1+K])
              break;
          if (K == DLenEnd)
           {
            /* estacao ja conectada */
            printf("************* usuario ja conectado na estacao \n");
            SESaux[0] = 5;
            break;
           }
         }
      /* checar senha */

/* colocar verificacao de senha */

      /* verifica se nao houve erro */
      if (SESaux[0] == 0)
       {
        /* procura uma sessao disponivel */
        for (I=1; I<=DMaxSes; I++)
          if ((int) TAB_SES[I].COD_USU == 0)
            break;
        /* verifica se ha' sessao disponivel */
```

```
        if (I > DMaxUsu)
          /* sessao nao disponivel */
          SESaux[0] = 1;
        else
         {
          /* sessao disponivel */
       printf("************* sessao disponivel = %d\n",I);
          SESaux[1] = (BYTE) I;
          TAB_SES[I].COD_USU = USUaux;
          for (K=0; K<DLenEnd; K++)
            TAB_SES[I].END_EST[K] = BUFFER[Str20+1+K];
          TAB_SES[I].IDE_PROC[0] = BUFFER[Str20+7];
          TAB_SES[I].IDE_PROC[1] = BUFFER[Str20+8];
         }
       }
     }
    BUFFER[0] = SESaux[0];
    BUFFER[1] = SESaux[1];
    *TAM_BUFFER = 2;
    printf("   Usuario = %02d\n",TAB_SES[I].COD_USU);
    printf(" Endereco = ");
    for (J=0; J<DLenEnd; J++)
      printf ("%02x ",TAB_SES[I].END_EST[J]);
    printf("\n   Sessao = %02x%02x\n",
         BUFFER[0],BUFFER[1]);
    printf("   Processo = %02x%02x\n",
         TAB_SES[I].IDE_PROC[0],TAB_SES[I].IDE_PROC[1]);
    printf("*** fim do login ***\n\n");
    return(0);
   }
```

12- DestroiSessão: finaliza a sessão do cliente no servidor.

```
int DestroiSessao(int *TAM_BUFFER, BYTE BUFFER[])
  {
   BYTE *aux;
   int SESSAOi;
   aux = (BYTE *) &SESSAOi;
   *aux = BUFFER[0];
   *(aux+1) = BUFFER[1];
   BUFFER[0] = 0x00;
   printf("*** inicio do logout ***\n");
   printf("\n   Sessao = %04d\n",SESSAOi);
   if (TAB_SES[SESSAOi].COD_USU == 0x00)
     BUFFER[0] = 0x01;
    else
     TAB_SES[SESSAOi].COD_USU = 0x00;
   *TAM_BUFFER = 1;
   printf("*** fim do logout ***\n");
   return(0);
  }
```

13- AnalisarPacote: verifica seqüencia, permissão, protocolo e sessão.

```
void AnalisarPacote(BYTE END_DST[6], BYTE END_ORG[6],
                    BYTE TIPO_PKT[2],
                    BYTE PROT, BYTE SESSAO[2],
                    BYTE PRIM, BYTE ESTADO,
                    BYTE DADOS[], int TAM_DADOS)
 {
  int INDICE, ERRO, NADA, PROTaux, SESSAOi, USUARIO;
  BYTE ESTADOaux, *aux, PROCESSO[2];
  int TAM_DADOSaux, J;
  BYTE DADOSaux[3000];
  div_t NumPacotes;
  int i;
  TAM_DADOSaux = TAM_DADOS;
  for (INDICE=0; INDICE<TAM_DADOSaux; INDICE++)
    DADOSaux[INDICE] = DADOS[INDICE];


/*
printf("Logo apos entrar em AnalisarPacote. TAm= %d\n",TAM_DADOSaux);
for (INDICE=0; INDICE<TAM_DADOSaux; INDICE++)
  printf("%02x ",DADOSaux[INDICE]);
*/

  PROTaux = (int) (PROT & 0x7f);
  ESTADOaux = 0x00;
  PROCESSO[0] = 0x01;
  PROCESSO[1] = 0x01;
  /* verifica protocolo */
  if (PROTaux != SPP)
   {
    ESTADOaux = ESTADOaux | 0x01;
   }
  else
   {
    /* verifica se e requisicao */
    if (PROT & (BYTE) 0x80)
     {
      ESTADOaux = ESTADOaux | 0x10;
     }
    else
     {
      /* verifica sessao */
      aux = (BYTE *) &SESSAOi;
      *aux = SESSAO[0];
      *(aux+1) = SESSAO[1];
      USUARIO = TAB_SES[SESSAOi].COD_USU;
      if (USUARIO == 0)
       {
        ESTADOaux = ESTADOaux | 0x04;
       }
      else
       {
        /* recupera identificacao do processo */
```

```
/*          aux = (BYTE *) &(TAB_SES[SESSAOi].IDE_PROC);
            PROCESSO[0] = *aux;
            PROCESSO[1] = *(aux+1);  */
            /* verifica primitiva */
          /*  if ()
              ESTADOaux = ESTADOaux | 0x08;
            else                    */
              /* verifica permissao */
/*            if (! TAB_USU[USUARIO].PERMISSAO[PRIM])
                ESTADOaux = ESTADOaux | 0x08;  */
          }
        }
      }
     if (ESTADOaux)
      {
       /* transmitir erro */
/*     ESTADOaux = (TAB_SES[SESSAOi].SEQUENCIA[PRIM] & (BYTE) 0x80) |
              ESTADOaux; */
       ERRO = TransmitirPacoteSPP_toCLT(END_ORG,END_DST,
                                TIPO_PKT,
                                PROT,PROCESSO,
                                PRIM,ESTADOaux,
                                  "",0);
       if (ERRO)
         printf("pacote com estado 1 nao foi transmitido.\n");
      }
     else
      {
       /* verifica sequencia */
       if ((TAB_SES[SESSAOi].SEQUENCIA[PRIM] &
         (BYTE) 0x80) ==
         (ESTADO & (BYTE) 0x80))
        {
         /* transmite pacote anterior */
         for (INDICE=0; INDICE<TAM_DADOSaux; INDICE++)
            DADOSaux[INDICE] = TAB_SES[SESSAOi].RESP[INDICE];
            TAM_DADOSaux = TAB_SES[SESSAOi].TAM_RESP;
         ERRO = TransmitirPacoteSPP_toCLT(END_ORG,END_DST,
                                TIPO_PKT,
                                PROT,PROCESSO,
                                PRIM,ESTADOaux,
                                DADOSaux,
                                TAM_DADOSaux);
         if (ERRO)
            printf("pacote anterior nao foi transmitido.\n");
        }
       else
        {
         /* executar procedimento */
         ERRO = (*Primitiva[PRIM])(&TAM_DADOS,DADOS);

/*
printf("apos executar primitiva\n");
for (i=0; i<TAM_DADOSaux; i++)
  printf("%02x ",DADOSaux[i]);
printf("\n");
```

```
printf("tamanho: %d\n",TAM_DADOSaux);
*/
        if (ERRO)
         {
          /* transmitir erro */
          ESTADOaux = ESTADOaux | 0x20;
          ERRO = TransmitirPacoteSPP_toCLT(END_ORG,END_DST,
                                TIPO_PKT,
                                PROT,PROCESSO,
                                PRIM,ESTADOaux,
                                "",0);
          if (ERRO)
            printf("pacote com erro na primitiva nao foi transmitido.\n");
         }
        else
         {
          /* transmitir resposta */
          TAB_SES[SESSAOi].SEQUENCIA[PRIM] =
               (BYTE) (~TAB_SES[SESSAOi].SEQUENCIA[PRIM]);
          ESTADOaux = TAB_SES[SESSAOi].SEQUENCIA[PRIM] & (BYTE) 0x80;
          for (INDICE=0; INDICE<TAM_DADOSaux; INDICE++)
            TAB_SES[SESSAOi].RESP[INDICE] = DADOSaux[INDICE];
            TAB_SES[SESSAOi].TAM_RESP = TAM_DADOSaux;

            /* 1490 = TAM. DO PKT - 24 */
            NumPacotes = div (TAM_DADOS, 1490);
            if ((NumPacotes.rem) || (TAM_DADOS == 0))  NumPacotes.quot++;
            strncpy(DADOSaux, (BYTE *) &(NumPacotes.quot), 2);
            INDICE = 0;
             while ((INDICE < TAM_DADOS) || (INDICE==0)){
                  TAM_DADOSaux = 2;
                  while ((TAM_DADOSaux < 1490) && (INDICE < TAM_DADOS)){
                        DADOSaux[TAM_DADOSaux] = DADOS[INDICE];
                        TAM_DADOSaux++;
                        INDICE++;
                       } /* Fim while 1 */
                  if ((INDICE == 0) && (TAM_DADOS == 0)) INDICE++;
                  ERRO = TransmitirPacoteSPP_toCLT(END_ORG,END_DST,
                              TIPO_PKT,
                              PROT,PROCESSO,
                              PRIM,ESTADOaux,
                              DADOSaux,TAM_DADOSaux);
                  if (ERRO)
                     printf("pacote OK nao foi transmitido.\n");
          }}
       }
     }
```

14- AtenderProcedimentoRemoto: recebe a requisição e executa a primitiva solicitada.

```
void AtenderProcedimentoRemoto(void) {
  int I, ERRO, NADA;
```

```
    BYTE RecBuf[DLenPacote], END_DST[6], END_ORG[6], TIPO_PKT[2],
     PROT, SESSAO[2], PRIM, ESTADO, PROCESSO[2];
    static BYTE DADOS[1][3000];
    int TamRecBuf, NumPacotes[5], TamDados[5], SESSAOi;
    char CH;
    BYTE *aux;
    CH = ' ';
    for (I=0; I<5; I++) {
      NumPacotes[I]=0;
/*    DADOS[I] = (BYTE *) malloc (3000);       */
    }
    for (;;) {
      /* atendimento de solicitacoes dos cliente ate' que a tecla <@>
         seja digitada, ou seja, <shift 2> seja digitada */
      if (kbhit())
       {
        CH = (char) getch();
        if (CH == '@')
         {
          for (I=0;I<DMaxSes;I++)
           {
            ERRO = TerminarAcessoTipoPacote(GInt,GHandle[I]);
            if (ERRO)
              ImprimeErro(404);
           }
          exit(0);
         }
       }
      /* verifica se chegou algum pacote */
      if (ReceberPacoteSPP_fromCLT(END_DST,END_ORG,TIPO_PKT,
                          &PROT,SESSAO,&PRIM,&ESTADO,
       RecBuf,&TamRecBuf)) {

/*
      printf("Mens %d- TamRecBuf=%d\nBufRec:",++cont,TamRecBuf);
      for (I=0; I<TamRecBuf; I++)
        printf("%02x ",RecBuf[I]);
      printf("\n");
*/
     if ((PRIM != LOG_INN) && (PRIM != LOG_OUT)){
       SESSAOi = (int) SESSAO[0] - 1;
       if (NumPacotes[SESSAOi] == 0){
         aux = (BYTE*) &NumPacotes[SESSAOi];
        *aux = RecBuf[0];
        *(aux+1) = RecBuf[1];
/*       DADOS[SESSAOi] = (BYTE *) malloc (NumPacotes[SESSAOi] * 1490);*/
        TamDados[SESSAOi] = 0;
       }
       for (I=2; I<TamRecBuf; I++){
        DADOS[SESSAOi][TamDados[SESSAOi]]=RecBuf[I];
        TamDados[SESSAOi]++;
       }
/*
       printf("Imprime Dados depois de copiado\n");
       for (I=0; I<TamDados[SESSAOi]; I++)
         printf("%02x ",DADOS[SESSAOi][I]);
```

```
            printf("\n");
*/
          NumPacotes[SESSAOi]--;
          if (NumPacotes[SESSAOi] != 0)
            /* Recebeu parte dos parametros */
            EnviaAck(END_DST,END_ORG, TIPO_PKT, PROT,SESSAO, PRIM);
          else{
            AnalisarPacote(END_DST,END_ORG, TIPO_PKT, PROT,SESSAO,
           PRIM,ESTADO, DADOS[SESSAOi],TamDados[SESSAOi]);
/*          free(DADOS[SESSAOi]); */
           }
        }else
          if (PRIM == LOG_INN)
            AnalisarLogin(END_DST,END_ORG, TIPO_PKT, PROT,
                          PRIM,ESTADO, RecBuf,TamRecBuf);
            else
              AnalisarLogout(END_DST,END_ORG, TIPO_PKT, PROT,SESSAO,
                             PRIM,ESTADO, RecBuf,TamRecBuf);
          }
    }/* End for(;;) */
}
```

15- CarregaUsuarios: inicia a estrutura de dados responsável pelas informações sobre os usuarios conectados.

```
/* carrega o arquivo de usuarios na tabela de usuarios */
void CarregaUsuarios(void)
 {
  int I, J;
  FILE *ARQ_USU;
  char PALAVRA[Str20] = "\0";
  char CAMINHO[Str20] = "\\gaps\\\0";
  char NOME_ARQ[Str40] = "\0";
  strcpy(NOME_ARQ,CAMINHO);
  strcat(NOME_ARQ,"USUARIOS");
  if ((ARQ_USU=fopen(NOME_ARQ,"r")) == NULL)
   {
    ImprimeErro(51);
    exit(0);
   }
  for (I=1; I<=DMaxUsu; I++)
   {
    fscanf(ARQ_USU,"%10s",PALAVRA);
    for (J=0; PALAVRA[J]!='\0' && PALAVRA[J]!=' '
          && PALAVRA[J]!='\n'; J++)
      PALAVRA[J] = toupper(PALAVRA[J]);
    PALAVRA[J] = '\0';
    strcpy(TAB_USU[I].NOME_USU,PALAVRA);
    fscanf(ARQ_USU,"%2d",&(TAB_USU[I].SENHA));
    for (J=0; J<DMaxPrim; J++)
      TAB_USU[I].PERMISSAO[J]= 0xff;
   }
  fclose(ARQ_USU);
 }
```

16- CarregaSessões: inicia a estrutura de dados responsável pelas informações sobre as sessões.

```
void CarregaSessoes(void)
  {
   int I, J;
   for (I=1;I<=DMaxSes;I++)
     {
      /* Iniciar o codigo do usuario de todas as sessoes com zero */
      TAB_SES[I].COD_USU = 0x00;
      /* Iniciar as sequencias */
      for (J=0; J<DMaxPrim; J++)
        TAB_SES[I].SEQUENCIA[J]= 0xff;
     }
  }
```

APÊNDICE B

Códigos de implementação das funções do sistema GAPS

Neste apêndice estão listadas as rotinas do gerador de *stubs*, implementadas em turbo-C:

Analisador Léxico

```
void AnalisadorLexico()
  {
   char LINHA[LEN_LINHA];
   char ATOMO[LEN_ATOMO];
   int  NUM_LINHA = 1;
   int  LINHA_COMENT = 0;
   int  LINHA_CADEIA = 0;
   int  LINHA_CARACTER = 0;
   int  PROXIMO = 0;
   int  INDICE = 0;
   enum BOLEANO COMENT = FALSO;
   enum BOLEANO IDENT = FALSO;
   enum BOLEANO NUM = FALSO;
   enum BOLEANO CAD = FALSO;
   enum BOLEANO CARAC = FALSO;
   ATOMO[0] = EOS;
   fgets(LINHA,LEN_LINHA,ARQdef);
   while (! feof(ARQdef))
    {
     /* fim de string */
     if (LINHA[PROXIMO] == EOS)
      {
       PROXIMO = 0;
       fgets(LINHA,LEN_LINHA,ARQdef);
      }
     else
       /* captura identificador ou palavra reservada */
       if (IDENT)
        {
         if ((isalnum(LINHA[PROXIMO])) ||
```

```
    (LINHA[PROXIMO] == UNDERSCORE) ||
    (LINHA[PROXIMO] == ABRECOLCH) ||
    (LINHA[PROXIMO] == FECHACOLCH))
   {
    ATOMO[INDICE++] = LINHA[PROXIMO++];
   }
   else
    if (LINHA[PROXIMO] != EOS)
     {
      ATOMO[INDICE] = EOS;
      if (PalavraReservada(ATOMO))
        InserirToken(ATOMO,RESERVADA,NUM_LINHA);
       else
        if (IdentOk(ATOMO))
          InserirToken(ATOMO,IDENTIFICADOR,NUM_LINHA);
         else
          {
           ImprimeErro(NUM_LINHA,13,ATOMO);
          }
      IDENT = FALSO;
      INDICE = 0;
     }
  }
  else
   /* captura numero */
   if (NUM)
    {
     if ((isdigit(LINHA[PROXIMO])) ||
       (LINHA[PROXIMO] == PONTO))
      {
       ATOMO[INDICE++] = LINHA[PROXIMO++];
      }
      else
       if (LINHA[PROXIMO] != EOS)
        {
         ATOMO[INDICE] = EOS;
         if (NumOk(ATOMO))
           InserirToken(ATOMO,NUMERO,NUM_LINHA);
          else
           {
            ImprimeErro(NUM_LINHA,14,ATOMO);
           }
         NUM = FALSO;
         INDICE = 0;
        }
   }
   else
    /* fim de linha */
    if (LINHA[PROXIMO] == EOL)
     {
      NUM_LINHA++;
      PROXIMO = 0;
      fgets(LINHA,LEN_LINHA,ARQdef);
     }
     else
      /* captura cadeia */
```

```
if (CAD)
  if (LINHA[PROXIMO] == ASPAS_DUPLA)
   {
    ATOMO[INDICE] = EOS;
    InserirToken(ATOMO,CADEIA,NUM_LINHA);
    PROXIMO++;
    CAD = FALSO;
    INDICE = 0;
   }
  else
   {
    if (INDICE == LEN_ATOMO-2)
     {
      ImprimeErro(LINHA_CADEIA,20,EOS);
      CAD = FALSO;
      INDICE = 0;
     }
    else
     {
      ATOMO[INDICE++] = LINHA[PROXIMO++];
     }
   }
else
 /* captura caracter */
 if (CARAC)
  {
   if (LINHA[PROXIMO] == ASPAS_SIMPLES)
    {
     ATOMO[INDICE] = EOS;
     InserirToken(ATOMO,CARACTER,NUM_LINHA);
     PROXIMO++;
     CARAC = FALSO;
     INDICE = 0;
    }
   else
    {
     if (INDICE == 0)
      {
       ATOMO[INDICE++] = LINHA[PROXIMO++];
      }
     else
      {
       ImprimeErro(LINHA_CARACTER,19,EOS);
       CARAC = FALSO;
       INDICE = 0;
      }
    }
  }
 else
  /* elimina comentario */
  if (COMENT)
    if (LINHA[PROXIMO] == ASTERISCO)
      if (LINHA[++PROXIMO] == BARRA)
       {
        PROXIMO++;
        COMENT = FALSO;
```

```
        }
       else
         PROXIMO++;
     else
      PROXIMO++;
  else
  /* captura simbolo especial */
  if (SimboloEspecial(LINHA[PROXIMO]))
   {
    ATOMO[INDICE++] = LINHA[PROXIMO++];
    ATOMO[INDICE] = EOS;
    InserirToken(ATOMO,ESPECIAL,NUM_LINHA);
    INDICE = 0;
   }
  else
   /* verifica inicio de identificador ou
      palavra reservada */
   if ((isalpha(LINHA[PROXIMO])) ||
      (LINHA[PROXIMO] == ASTERISCO) ||
      (LINHA[PROXIMO] == UNDERSCORE) ||
      (LINHA[PROXIMO] == CANCELA))
    {
     IDENT = VERDADEIRO;
     ATOMO[INDICE++] = LINHA[PROXIMO++];
    }
    else
     /* verifica inicio de numero */
     if (isdigit(LINHA[PROXIMO]))
      {
       NUM = VERDADEIRO;
       ATOMO[INDICE++] = LINHA[PROXIMO++];
      }
      else
       /* verifica inicio de cadeia */
       if (LINHA[PROXIMO] == ASPAS_DUPLA)
        {
         LINHA_CADEIA = NUM_LINHA;
         CAD = VERDADEIRO;
         PROXIMO++;
        }
        else
         /* verifica inicio de caracter */
         if (LINHA[PROXIMO] == ASPAS_SIMPLES)
          {
           LINHA_CARACTER = NUM_LINHA;
           CARAC = VERDADEIRO;
           PROXIMO++;
          }
          else
           /* elimina brancos */
           if (LINHA[PROXIMO] == BRANCO)
             PROXIMO++;
            else
             /* verifica inicio de comentario */
             if (LINHA[PROXIMO] == BARRA)
               {
```

```
                                    PROXIMO++;
                                    if (LINHA[PROXIMO] == ASTERISCO)
                                     {
                                      LINHA_COMENT = NUM_LINHA;
                                      COMENT = VERDADEIRO;
                                      PROXIMO++;
                                     }
                                     else
                                     {
                                      ImprimeErro(NUM_LINHA,11,EOS);
                                     }
                                   }
                                   else
                                   {
                                    ATOMO[INDICE++] = LINHA[PROXIMO++];
                                    ATOMO[INDICE] = EOS;
                                    INDICE = 0;
                                    ImprimeErro(NUM_LINHA,12,ATOMO);
                                   }
     }
    if (COMENT)
     {
      ImprimeErro(LINHA_COMENT,15,EOS);
     }
     else
      if (CAD)
       {
        ImprimeErro(LINHA_CADEIA,17,EOS);
       }
       else
        if (CARAC)
         {
          ImprimeErro(LINHA_CARACTER,18,EOS);
         }
  }
```

Analisador Sintático

```
enum BOLEANO Bloco()
  {
   enum BOLEANO ENTROU;
   while ((PTR_TOKEN != NULL) &&
        (! StrIgualChr(PTR_TOKEN->ATOMO,ESPECIAIS[FECHA_CHAVE])))
    {
     if (DefinicaoDeDefinicao())
      {
/*     Proximo(1); */
      }
      else
       if (DefinicaoDeConstante())
        {
         if (StrIgualChr(PTR_TOKEN->ATOMO,ESPECIAIS[PONTO_VIRG]))
           Proximo(1);
```

```
            else
            {
             /* ';' esperado */
             ImprimeErro(PTR_TOKEN->LINHA,38,PTR_TOKEN->ATOMO);
            }
          }
          else
           if (DefinicaoDeEnumeracao())
            {
             if (StrIgualChr(PTR_TOKEN->ATOMO,ESPECIAIS[PONTO_VIRG]))
               Proximo(1);
              else
              {
               /* ';' esperado */
               ImprimeErro(PTR_TOKEN->LINHA,38,PTR_TOKEN->ATOMO);
              }
            }
           else
            if (DefinicaoDeEstrutura())
             {
              if (StrIgualChr(PTR_TOKEN->ATOMO,ESPECIAIS[PONTO_VIRG]))
                Proximo(1);
               else
               {
                /* ';' esperado */
                ImprimeErro(PTR_TOKEN->LINHA,38,PTR_TOKEN->ATOMO);
               }
             }
             else
              if (DefinicaoDeSubrotina())
               {
                if (StrIgualChr(PTR_TOKEN->ATOMO,ESPECIAIS[PONTO_VIRG]))
                  Proximo(1);
                 else
                 {
                  /* ';' esperado */
                  ImprimeErro(PTR_TOKEN->LINHA,38,PTR_TOKEN->ATOMO);
                 }
               }
              else
               {
                /* item nao esperado */
                ImprimeErro(PTR_TOKEN->LINHA,37,PTR_TOKEN->ATOMO);
                ProximoPontoEVirgula();
                Proximo(1);
               }
   }
  if (HOUVE_ERRO)
    return(FALSO);
   else
    return(VERDADEIRO);
 }

void Definition()
 {
```

```
PTR_TOKEN = LISTA_TOKEN;
if (strcmp(PTR_TOKEN->ATOMO,RESERVADAS[DEFINITION]))
 {
  /* erro na estrutura de definicao */
  ImprimeErro(PTR_TOKEN->LINHA,31,EOS);
  /* 'definition' esperado */
  ImprimeErro(PTR_TOKEN->LINHA,32,EOS);
 }
else
 {
  Proximo(1);
  if (PTR_TOKEN->CLASSE != IDENTIFICADOR)
   {
    /* erro na estrutura de definicao */
    ImprimeErro(PTR_TOKEN->LINHA,31,EOS);
    /* identificador do modulo esperado */
    ImprimeErro(PTR_TOKEN->LINHA,33,EOS);
   }
  else
   {
    Proximo(1);
    if (! StrIgualChr(PTR_TOKEN->ATOMO,ESPECIAIS[ABRE_CHAVE]))
     {
      /* erro na estrutura de definicao */
      ImprimeErro(PTR_TOKEN->LINHA,31,EOS);
      /* '{' esperado ==> inicio de modulo de definicao */
      ImprimeErro(PTR_TOKEN->LINHA,34,EOS);
     }
    else
     {
      Proximo(1);
      if (Bloco())
       {
        if (! StrIgualChr(PTR_TOKEN->ATOMO,ESPECIAIS[FECHA_CHAVE]))
         {
          /* erro na estrutura de definicao */
          ImprimeErro(ULTIMO_TOKEN->LINHA,31,EOS);
          /* '}' esperado ==> fim de modulo de definicao */
          ImprimeErro(ULTIMO_TOKEN->LINHA,35,EOS);
         }
        else
         {
          Proximo(1);
          if (PTR_TOKEN != NULL)
           {
            /* erro na estrutura de definicao */
            ImprimeErro(PTR_TOKEN->LINHA,31,EOS);
            /* informacao apos fim de modulo de definicao */
            ImprimeErro(PTR_TOKEN->LINHA,36,EOS);
           }
         }
       }
     }
   }
 }
}
```

```
void AnalisadorSintatico()
  {
   printf("Analisador Sintatico:\n");
   Definition();
  }
```

Analisador Semântico

```
void Verificacoes()
  {
   struct SIMBOLO *SIMBaux;
   SIMBaux  = LISTA_SIMBOLO;
   while (SIMBaux != NULL)
     {
      switch(SIMBaux->TIPO)
       {
        case CON:
            break;
        case ENU:
            /* verifica se existe constante enumeracao duplicada */
            VerificaConstEnum(SIMBaux->ATOMO);
            break;
        case EST:
            /* verifica se existe variavel duplicada */
            /* verifica se tipo ja definido */
            /* marca os tipos e constantes que usou */
            VerificaCampos(SIMBaux->ATOMO);
            break;
        case FUN:
            VerificaTipo(SIMBaux->ATOMO);
            /* verifica se existe argumento duplicado */
            /* verifica se tipo ja definido */
            /* marca os tipos e constantes que usou */
            VerificaArgumentos(SIMBaux->ATOMO);
            break;
        case PRO:
            /* verifica se existe argumento duplicado */
            /* verifica se tipo ja definido */
            /* marca os tipos e constantes que usou */
            VerificaArgumentos(SIMBaux->ATOMO);
            break;
       }
      SIMBaux  = SIMBaux->PROXIMO;
     }
  }


void AnalisadorSemantico()
  {
   printf("Analisador Semantico:\n");
   Verificacoes();
   if (! HOUVE_ERRO) QuaisSimbolosNaoUsou();
```

```
  }
```

Gerador

```
void EscrevePrimitivas()
  {
   int CONTADOR;
   fprintf(HEADER,
     " /* codigos de identificacao dos procedimentos (primitivas) */\n");
   for (CONTADOR=1; CONTADOR<=NUM_PROCS; CONTADOR++)
    {
/*    fprintf(HEADER," #define PROC_%02d   0x%02x\n",CONTADOR,
          PRIMITIVA[CONTADOR-1]); */
     fprintf(HEADER," #define PROC_%02d   0x%02x\n",CONTADOR,
          CONTADOR+1);
    }
  }


void EscrevePrototipos()
 {
  struct SUBROTINA *SUBaux;
  struct ARGUMENTO *ARGaux;
  struct INDICE *INDaux;
  fprintf(HEADER,"\n /* prototipos dos procedimentos */\n");
  SUBaux = LISTA_SUBROTINA;
  while (SUBaux != NULL)
   {
    if (strcmp(SUBaux->TIPO,RESERVADAS[VOID]))
      if (SUBaux->ENUM_OU_ESTR == ENU)
        fprintf(HEADER," enum %s ",SUBaux->TIPO);
       else
        if (SUBaux->ENUM_OU_ESTR == EST)
          fprintf(HEADER," struct %s ",SUBaux->TIPO);
         else
          if (strcmp(SUBaux->TIPO,RESERVADAS[STRING]))
            fprintf(HEADER," %s ",SUBaux->TIPO);
           else
            fprintf(HEADER," char *",SUBaux->TIPO);
    else
      fprintf(HEADER," %s ",SUBaux->TIPO);
    fprintf(HEADER,"%s(",SUBaux->NOME);
    ARGaux = SUBaux->PRIM_ARG;
    while (ARGaux != NULL)
     {
      /* escreve os tipos dos argumentos */
      if (ARGaux->ENUM_OU_ESTR == ENU)
        fprintf(HEADER,"enum %s ",ARGaux->TIPO);
       else
        if (ARGaux->ENUM_OU_ESTR == EST)
          fprintf(HEADER,"struct %s ",ARGaux->TIPO);
         else
          if (strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
            fprintf(HEADER,"%s ",ARGaux->TIPO);
           else
            fprintf(HEADER,"char ");
```

```
        /* escreve os nomes dos argumentos */
        if (! strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
          if (ARGaux->PRIM_IND != NULL)
            fprintf(HEADER,"%s[%s][]",ARGaux->VARIAVEL,
                 ARGaux->PRIM_IND->IND);
           else
            fprintf(HEADER,"%s[]",ARGaux->VARIAVEL,
                 ARGaux->PRIM_IND->IND);
         else
          if (ARGaux->PRIM_IND != NULL)
            fprintf(HEADER,"%s[]",ARGaux->VARIAVEL);
           else
            if (ARGaux->PARAMETRO == IN)
              fprintf(HEADER,"%s",ARGaux->VARIAVEL);
             else
              fprintf(HEADER,"*%s",ARGaux->VARIAVEL);
        ARGaux = ARGaux->PROXIMO;
        if (ARGaux != NULL)
          fprintf(HEADER,",");
       }
     fprintf(HEADER,");\n");
     SUBaux = SUBaux->PROXIMO;
     }
  }


void CriaHeader()
 {
  int CONTADOR;
  struct CONSTANTE *CONSaux;
  struct ENUMERACAO *ENUMaux;
  struct CONS_ENUM *CONS_ENUMaux;
  struct ESTRUTURA *ESTRaux;
  struct CAMPO_ESTR *CAMPOaux;
  struct DEFINICAO *DEFaux;
  struct INDICE *INDaux;
  DEFaux = LISTA_DEFINICAO;
  while (DEFaux != NULL)
   {
    fprintf(HEADER,
          " #define %s %s \n",DEFaux->NOME,DEFaux->VALOR);
    DEFaux = DEFaux->PROXIMO;
   }
  fprintf(HEADER," #define LEN_MAX_STR   256\n\n");
  fprintf(HEADER," /* definicao da variavel do tipo union */\n");
  fprintf(HEADER," union TIPO_PRE TAM_TIPO_PRE;\n\n");
  EscrevePrimitivas();
  /* escreve as declaracoes de constantes */
  CONSaux = LISTA_CONSTANTE;
  if (CONSaux != NULL)
   {
    fprintf(HEADER,"\n /* constantes */\n");
    while (CONSaux != NULL)
     {
      fprintf(HEADER," const int %s = %s;\n",CONSaux->NOME,
           CONSaux->VALOR);
```

```
      CONSaux = CONSaux->PROXIMO;
     }
   }
  /* escreve as declaracoes de enumeracoes */
  ENUMaux = LISTA_ENUMERACAO;
  if (ENUMaux != NULL)
   {
    fprintf(HEADER,"\n /* enumeracoes */\n");
    while (ENUMaux != NULL)
     {
      fprintf(HEADER," enum %s { ",ENUMaux->NOME);
      CONS_ENUMaux = ENUMaux->PRIM_CONS;
      while (CONS_ENUMaux != NULL)
       {
        fprintf(HEADER,"%s = %3d",CONS_ENUMaux->NOME,
              CONS_ENUMaux->VALOR);
        CONS_ENUMaux = CONS_ENUMaux->PROXIMO;
        if (CONS_ENUMaux != NULL)
         {
          fprintf(HEADER,",\n        ");
          for (CONTADOR=1; CONTADOR<=strlen(ENUMaux->NOME); CONTADOR++)
           {
            fprintf(HEADER," ");
           }
         }
        else
         {
          fprintf(HEADER," };\n");
         }
       }
      ENUMaux = ENUMaux->PROXIMO;
     }
   }
  /* escreve as declaracoes de estruturas */
  ESTRaux = LISTA_ESTRUTURA;
  if (ESTRaux != NULL)
   {
    fprintf(HEADER,"\n /* estruturas */\n");
    while (ESTRaux != NULL)
     {
      fprintf(HEADER," struct %s { ",ESTRaux->NOME);
      CAMPOaux = ESTRaux->PRIM_CAMPO;
      while (CAMPOaux != NULL)
       {
        if (CAMPOaux->ENUM_OU_ESTR == ENU)
         {
          fprintf(HEADER,"enum %s  ",CAMPOaux->TIPO);
         }
        else
         if (CAMPOaux->ENUM_OU_ESTR == EST)
          {
           fprintf(HEADER,"struct %s  ",CAMPOaux->TIPO);
          }
         else
          {
           if (strcmp(CAMPOaux->TIPO,RESERVADAS[STRING]))
```

```
                {
                 fprintf(HEADER,"%s ",CAMPOaux->TIPO);
                }
               else
                {
                 fprintf(HEADER,"char *");
                }
              }
            fprintf(HEADER,"%s",CAMPOaux->VARIAVEL);
            INDaux = CAMPOaux->PRIM_IND;
            while (INDaux != NULL)
             {
              fprintf(HEADER,"[%s]",INDaux->IND);
              INDaux = INDaux->PROXIMO;
             }
            fprintf(HEADER,";",CAMPOaux->VARIAVEL);
            CAMPOaux = CAMPOaux->PROXIMO;
            if (CAMPOaux != NULL)
             {
              fprintf(HEADER,"\n          ");
              for (CONTADOR=1; CONTADOR<=strlen(ESTRaux->NOME); CONTADOR++)
               {
                fprintf(HEADER," ");
               }
             }
            else
             {
              fprintf(HEADER," };\n");
             }
           }
          ESTRaux = ESTRaux->PROXIMO;
         }
        }
       EscrevePrototipos();
      }


     char *ProcuraTipo(char TIPOent[LEN_ATOMO])
      {
       if ((! strcmp(TIPOent,RESERVADAS[CHAR])) ||
          (! strcmp(TIPOent,RESERVADAS[UNS_CHAR])) ||
          (! strcmp(TIPOent,RESERVADAS[SIG_CHAR])))
          return("CHAR");
        else
          if ((! strcmp(TIPOent,RESERVADAS[INT])) ||
             (! strcmp(TIPOent,RESERVADAS[UNS_INT])) ||
             (! strcmp(TIPOent,RESERVADAS[SIG_INT])))
             return("INT");
           else
             if ((! strcmp(TIPOent,RESERVADAS[SHT_INT])) ||
                (! strcmp(TIPOent,RESERVADAS[UNS_SHT_INT])) ||
                (! strcmp(TIPOent,RESERVADAS[SIG_SHT_INT])))
                return("SINT");
              else
                if ((! strcmp(TIPOent,RESERVADAS[LONG_INT])) ||
                   (! strcmp(TIPOent,RESERVADAS[UNS_LONG_INT])) ||
```

```
            (! strcmp(TIPOent,RESERVADAS[SIG_LONG_INT]))
            return("LINT");
          else
            if (! strcmp(TIPOent,RESERVADAS[FLOAT]))
              return("FLOAT");
            else
              if (! strcmp(TIPOent,RESERVADAS[DOUBLE]))
                return("DOUB");
              else
                if (! strcmp(TIPOent,RESERVADAS[LONG_DOUBLE]))
                  return("LDOUB");
                else
                  return("");
}

int ProcuraEstrutura(char ESTRUT[LEN_ATOMO],char CAMPO[LEN_ATOMO],
                     char INDICE[LEN_ATOMO])
{
 struct ESTRUTURA *ESTRaux;
 struct CAMPO_ESTR *CAMPaux;
 ESTRaux = LISTA_ESTRUTURA;
 while ((ESTRaux != NULL) &&
        (strcmp(ESTRaux->NOME,ESTRUT)))
   ESTRaux = ESTRaux->PROXIMO;
 if (ESTRaux == NULL)
   return(0);
 else
  {
   CAMPaux = ESTRaux->PRIM_CAMPO;
   while ((CAMPaux != NULL) &&
          (strcmp(CAMPaux->VARIAVEL,CAMPO)))
     CAMPaux = CAMPaux->PROXIMO;
   if (ESTRaux == NULL)
     return(0);
    else
     if (CAMPaux->PRIM_IND == NULL)
       return(0);
      else
       {
        strcpy(INDICE,CAMPaux->PRIM_IND->IND);
        return(1);
       }
  }
}

void EscreveArgToMsg(FILE *ARQent, struct PARAMETRO *PARent, int ENT_SAI)
{
 char INDICE[LEN_ATOMO];
 char *TIPOaux;
 while (PARent != NULL)
  {
   INDICE[0] = EOS;
   TIPOaux = ProcuraTipo(PARent->TIPO);
   /* verifica se eh estrutura */
```

```
if (PARent->ENUM_OU_ESTR == EST)
 {
  /* verifica se o campo da estrutura tem indice */
  if (ProcuraEstrutura(PARent->TIPO_TP,PARent->VARIAVEL,INDICE))
   {
    fprintf(ARQent,"    for (J_1=0; J_1<%s; J_1++)\n",
          INDICE);
    fprintf(ARQent,"      {\n");
    /* verifica se a estrutura tem indice */
    if (PARent->PRIM_IND != NULL)
     {
      fprintf(ARQent,"       for (I_1=0; I_1<%s; I_1++)\n",
            PARent->PRIM_IND->IND);
      fprintf(ARQent,"         {\n");
      if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
        if (ENT_SAI == IN)
          fprintf(ARQent,
           "          strcpy(ARGUMENTO.STRING,%s[I_1].%s[J_1]);\n",
                PARent->NOME_TP,PARent->VARIAVEL);
         else
          fprintf(ARQent,
           "          strcpy(ARGUMENTO.STRING,%s[I_1].%s[J_1]);\n",
                PARent->NOME_TP,PARent->VARIAVEL);
       else
        if (ENT_SAI == IN)
          fprintf(ARQent,
           "          ARGUMENTO.%s = %s[I_1].%s[J_1];\n",TIPOaux,
                PARent->NOME_TP,PARent->VARIAVEL);
         else
          fprintf(ARQent,
           "          ARGUMENTO.%s = %s[I_1].%s[J_1];\n",TIPOaux,
                PARent->NOME_TP,PARent->VARIAVEL);
     }
    else
     {
      if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
        if (ENT_SAI == IN)
          fprintf(ARQent,
           "       strcpy(ARGUMENTO.STRING,%s.%s[J_1]);\n",
           PARent->NOME_TP,PARent->VARIAVEL);
         else
          fprintf(ARQent,
           "       strcpy(ARGUMENTO.STRING,%s->%s[J_1]);\n",
           PARent->NOME_TP,PARent->VARIAVEL);
       else
        if (ENT_SAI == IN)
          fprintf(ARQent,
           "       ARGUMENTO.%s = %s.%s[J_1];\n",TIPOaux,
           PARent->NOME_TP,PARent->VARIAVEL);
         else
          fprintf(ARQent,
           "       ARGUMENTO.%s = %s->%s[J_1];\n",TIPOaux,
           PARent->NOME_TP,PARent->VARIAVEL);
     }
   }
 else
```

```
    {
     if (PARent->PRIM_IND != NULL)
      {
       fprintf(ARQent,"    for (I_1=0; I_1<%s; I_1++)\n",
            PARent->PRIM_IND->IND);
       fprintf(ARQent,"      {\n");
       if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
         if (ENT_SAI == IN)
           fprintf(ARQent,"      strcpy(ARGUMENTO.STRING,%s[I_1].%s);\n",
                PARent->NOME_TP,PARent->VARIAVEL);
         else
           fprintf(ARQent,"      strcpy(ARGUMENTO.STRING,%s[I_1].%s);\n",
                PARent->NOME_TP,PARent->VARIAVEL);
       else
         if (ENT_SAI == IN)
           fprintf(ARQent,"      ARGUMENTO.%s = %s[I_1].%s;\n",TIPOaux,
                PARent->NOME_TP,PARent->VARIAVEL);
         else
           fprintf(ARQent,"      ARGUMENTO.%s = %s[I_1].%s;\n",TIPOaux,
                PARent->NOME_TP,PARent->VARIAVEL);
     }
     else
      if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
        if (ENT_SAI == IN)
          fprintf(ARQent,"    strcpy(ARGUMENTO.STRING,%s.%s);\n",
                PARent->NOME_TP,PARent->VARIAVEL);
        else
          fprintf(ARQent,"    strcpy(ARGUMENTO.STRING,%s->%s);\n",
                PARent->NOME_TP,PARent->VARIAVEL);
      else
        if (ENT_SAI == IN)
          fprintf(ARQent,"    ARGUMENTO.%s = %s.%s;\n",TIPOaux,
                PARent->NOME_TP,PARent->VARIAVEL);
        else
          fprintf(ARQent,"    ARGUMENTO.%s = %s->%s;\n",TIPOaux,
                PARent->NOME_TP,PARent->VARIAVEL);
    }
  }
  else
   if (PARent->ENUM_OU_ESTR == ENU)
    {
     if (ENT_SAI == IN)
       fprintf(ARQent,"    ARGUMENTO.INT = %s;\n",
            PARent->VARIAVEL);
     else
       fprintf(ARQent,"    ARGUMENTO.INT = *%s;\n",
            PARent->VARIAVEL);
    }
   else
    if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
     {
      if (PARent->PRIM_IND == NULL)
        fprintf(ARQent,"    strcpy(ARGUMENTO.STRING,%s);\n",
             PARent->VARIAVEL);
      else
       {
```

```
                fprintf(ARQent,"    for (I_1=0; I_1<%s; I_1++)\n",
                      PARent->PRIM_IND->IND);
                fprintf(ARQent,"      {\n");
                fprintf(ARQent,
                      "       strcpy(ARGUMENTO.STRING,%s[I_1]);\n",
                      PARent->VARIAVEL);
               }
              }
              else
               if (PARent->PRIM_IND != NULL)
                {
                fprintf(ARQent,"    for (I_1=0; I_1<%s; I_1++)\n",
                      PARent->PRIM_IND->IND);
                fprintf(ARQent,"      {\n");
                fprintf(ARQent,
                      "       ARGUMENTO.%s = %s[I_1];\n",
                      TIPOaux,PARent->VARIAVEL);
               }
               else
               {
                /* se parametro de entrada=IN ou retorno=0 */
                if ((ENT_SAI == IN) || (ENT_SAI == 0))
                  fprintf(ARQent,"    ARGUMENTO.%s = %s;\n",TIPOaux,
                       PARent->VARIAVEL);
                else
                  fprintf(ARQent,"    ARGUMENTO.%s = *%s;\n",TIPOaux,
                       PARent->VARIAVEL);
               }
        if ((PARent->PRIM_IND != NULL) && (INDICE[0] != EOS))
         {
          fprintf(ARQent,
           "          MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
           PARent->TIPO);
          fprintf(ARQent,"         };\n");
          fprintf(ARQent,"      };\n");
         }
        else
         if ((PARent->PRIM_IND != NULL) || (INDICE[0] != EOS))
          {
           fprintf(ARQent,
            "       MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
            PARent->TIPO);
           fprintf(ARQent,"      };\n");
          }
          else
           fprintf(ARQent,
            "    MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
            PARent->TIPO);
       PARent = PARent->PROXIMO;
      }
    }


void EscreveMsgToArg(FILE *ARQent, struct PARAMETRO *PARent, int ENT_SAI)
 {
  char INDICE[LEN_ATOMO];
```

```
char *TIPOaux;
while (PARent != NULL)
 {
  INDICE[0] = EOS;
  if (PARent->ENUM_OU_ESTR == EST)
    if (ProcuraEstrutura(PARent->TIPO_TP,PARent->VARIAVEL,INDICE))
      {
       fprintf(ARQent,"    for (J_1=0; J_1<%s; J_1++)\n",
             INDICE);
       fprintf(ARQent,"      {\n");
       if (PARent->PRIM_IND != NULL)
        {
         fprintf(ARQent,"       for (I_1=0; I_1<%s; I_1++)\n",
               PARent->PRIM_IND->IND);
         fprintf(ARQent,"         {\n");
         fprintf(ARQent,
                                                                    "
             DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
           PARent->TIPO);
        }
       else
        fprintf(ARQent,
                                                                    "
             DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
           PARent->TIPO);
      }
    else
      if (PARent->PRIM_IND != NULL)
       {
        fprintf(ARQent,"    for (I_1=0; I_1<%s; I_1++)\n",
             PARent->PRIM_IND->IND);
        fprintf(ARQent,"      {\n");
        fprintf(ARQent,
                                                                    "
             DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
           PARent->TIPO);
       }
       else
        fprintf(ARQent,
          "    DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
          PARent->TIPO);
  else
   if (PARent->PRIM_IND != NULL)
    {
     fprintf(ARQent,"    for (I_1=0; I_1<%s; I_1++)\n",
          PARent->PRIM_IND->IND);
     fprintf(ARQent,"      {\n");
     fprintf(ARQent,
      "       DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
      PARent->TIPO);
    }
    else
     fprintf(ARQent,
      "    DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,\"%s\");\n",
      PARent->TIPO);
 TIPOaux = ProcuraTipo(PARent->TIPO);
```

```
if (PARent->ENUM_OU_ESTR == EST)
  if (INDICE[0] != EOS)
   {
    /* verifica se a estrutura tem indice */
    if (PARent->PRIM_IND != NULL)
     {
      if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
        if (ENT_SAI == IN)
          fprintf(ARQent,
           "          strcpy(%s[I_1].%s[J_1],ARGUMENTO.STRING);\n",
                PARent->NOME_TP,PARent->VARIAVEL);
        else
          fprintf(ARQent,
           "          strcpy(%s[I_1].%s[J_1],ARGUMENTO.STRING);\n",
                PARent->NOME_TP,PARent->VARIAVEL);
      else
        if (ENT_SAI == IN)
          fprintf(ARQent,
           "          %s[I_1].%s[J_1] = ARGUMENTO.%s;\n",
                PARent->NOME_TP,PARent->VARIAVEL,TIPOaux);
        else
          fprintf(ARQent,
           "          %s[I_1].%s[J_1] = ARGUMENTO.%s;\n",
                PARent->NOME_TP,PARent->VARIAVEL,TIPOaux);
     fprintf(ARQent,"        };\n");
     fprintf(ARQent,"      };\n");
    }
    else
    {
     if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
       if (ENT_SAI == IN)
         fprintf(ARQent,
          "       strcpy(%s.%s[J_1],ARGUMENTO.STRING);\n",
          PARent->NOME_TP,PARent->VARIAVEL);
       else
         fprintf(ARQent,
          "       strcpy(%s->%s[J_1],ARGUMENTO.STRING);\n",
          PARent->NOME_TP,PARent->VARIAVEL);
      else
        if (ENT_SAI == IN)
          fprintf(ARQent,
           "       %s.%s[J_1] = ARGUMENTO.%s;\n",
           PARent->NOME_TP,PARent->VARIAVEL,TIPOaux);
        else
          fprintf(ARQent,
           "       %s->%s[J_1] = ARGUMENTO.%s;\n",
           PARent->NOME_TP,PARent->VARIAVEL,TIPOaux);
     fprintf(ARQent,"      };\n");
    }
  }
  else
  {
   if (PARent->PRIM_IND != NULL)
    {
     if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
       if (ENT_SAI == IN)
```

```
            fprintf(ARQent,"      strcpy(%s[I_1].%s,ARGUMENTO.STRING);\n",
                  PARent->NOME_TP,PARent->VARIAVEL);
           else
            fprintf(ARQent,"      strcpy(%s[I_1].%s,ARGUMENTO.STRING);\n",
                  PARent->NOME_TP,PARent->VARIAVEL);
         else
          if (ENT_SAI == IN)
            fprintf(ARQent,"      %s[I_1].%s = ARGUMENTO.%s;\n",
                  PARent->NOME_TP,PARent->VARIAVEL,TIPOaux);
           else
            fprintf(ARQent,"      %s[I_1].%s = ARGUMENTO.%s;\n",
                  PARent->NOME_TP,PARent->VARIAVEL,TIPOaux);
       fprintf(ARQent,"      };\n");
      }
      else
       if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
         if (ENT_SAI == IN)
           fprintf(ARQent,"   strcpy(%s.%s,ARGUMENTO.STRING);\n",
                  PARent->NOME_TP,PARent->VARIAVEL);
           else
            fprintf(ARQent,"   strcpy(%s->%s,ARGUMENTO.STRING);\n",
                  PARent->NOME_TP,PARent->VARIAVEL);
        else
         if (ENT_SAI == IN)
           fprintf(ARQent,"   %s.%s = ARGUMENTO.%s;\n",
                  PARent->NOME_TP,PARent->VARIAVEL,TIPOaux);
          else
           fprintf(ARQent,"   %s->%s = ARGUMENTO.%s;\n",
                  PARent->NOME_TP,PARent->VARIAVEL,TIPOaux);
   }
  else
   if (PARent->ENUM_OU_ESTR == ENU)
    {
     if (ENT_SAI == IN)
       fprintf(ARQent,"   %s = ARGUMENTO.INT;\n",
            PARent->VARIAVEL);
      else
       fprintf(ARQent,"   *%s = ARGUMENTO.INT;\n",
            PARent->VARIAVEL);
    }
   else
    if (! strcmp(PARent->TIPO,RESERVADAS[STRING]))
     {
      if (PARent->PRIM_IND == NULL)
        fprintf(ARQent,"   strcpy(%s,ARGUMENTO.STRING);\n",
             PARent->VARIAVEL);
       else
        {
         fprintf(ARQent,"      strcpy(%s[I_1],ARGUMENTO.STRING);\n",
              PARent->VARIAVEL);
         fprintf(ARQent,"      };\n");
        }
      }
     else
      if (PARent->PRIM_IND != NULL)
       {
```

```
            fprintf(ARQent,"      %s[I_1] = ARGUMENTO.%s;\n",
                  PARent->VARIAVEL,TIPOaux);
            fprintf(ARQent,"      };\n");
           }
           else
           {
            /* se parametro de entrada=IN ou retorno=0 */
            if ((ENT_SAI == IN) || (ENT_SAI == 0))
              fprintf(ARQent,"    %s = ARGUMENTO.%s;\n",
                    PARent->VARIAVEL,TIPOaux);
             else
              fprintf(ARQent,"    *%s = ARGUMENTO.%s;\n",
                    PARent->VARIAVEL,TIPOaux);
           }
    PARent = PARent->PROXIMO;
   }
 }


void CriaClientStub()
 {
  int CONTADOR = 0;
  struct SUBROTINA *SUBaux;
  struct ARGUMENTO *ARGaux;
  struct INDICE *INDaux;
  struct ELEMENTO_SUB *PROCaux;
  struct PARAMETRO *PARaux;
  enum OPCAO EXISTE_INDICE;
  fprintf(CLIENTE," extern int GNumPrim;\n\n");
  fprintf(CLIENTE," void CompletaTabPri()\n");
  fprintf(CLIENTE,"   {\n");
  fprintf(CLIENTE,"    int I;\n");
  /* carrega a tabela de primitivas */
  fprintf(CLIENTE,
        "    /* carrega a tabela de primitivas */\n");
  SUBaux = LISTA_SUBROTINA;
  PROCaux = TABELA_SUB;
  CONTADOR = 0;
  while (SUBaux != NULL)
   {
    CONTADOR++;
    fprintf(CLIENTE,"    AtribueTabPrim(%d,PROC_%02d,10,\"%s\\0\");\n",
          CONTADOR-1,CONTADOR,SUBaux->SERVIDOR);
    SUBaux = SUBaux->PROXIMO;
   }
  fprintf(CLIENTE,"    GNumPrim = %d;\n",NUM_PROCS);
  fprintf(CLIENTE,"   }\n\n");
  fprintf(CLIENTE," void Inicializa()\n");
  fprintf(CLIENTE,"   {\n");
  fprintf(CLIENTE,"    static int PRIM_VEZ = 1;\n");
  fprintf(CLIENTE,"    if (PRIM_VEZ)\n");
  fprintf(CLIENTE,"      {\n");
  fprintf(CLIENTE,"       PRIM_VEZ = 0;\n");
  fprintf(CLIENTE,"       CarregaServidores();\n");
  fprintf(CLIENTE,"       CarregaConfiguracao();\n");
  fprintf(CLIENTE,"       CompletaTabPri();\n");
```

```
fprintf(CLIENTE,"     }\n");
fprintf(CLIENTE,"  }\n\n");
SUBaux = LISTA_SUBROTINA;
PROCaux = TABELA_SUB;
CONTADOR = 0;
while (SUBaux != NULL)
 {
  CONTADOR++;
  /* escreve o tipo da funcao/procedimento */
  if (SUBaux->ENUM_OU_ESTR == ENU)
    fprintf(CLIENTE," enum %s ",SUBaux->TIPO);
   else
    if (SUBaux->ENUM_OU_ESTR == EST)
      fprintf(CLIENTE," struct %s ",SUBaux->TIPO);
     else
      if (strcmp(SUBaux->TIPO,RESERVADAS[STRING]))
        fprintf(CLIENTE," %s ",SUBaux->TIPO);
       else
        fprintf(CLIENTE," char *");
  /* escreve o nome da funcao/procedimento */
  fprintf(CLIENTE,"%s(",SUBaux->NOME);
  /* escreve os argumentos da funcao/procedimento */
  ARGaux = SUBaux->PRIM_ARG;
  while (ARGaux != NULL)
   {
    /* escreve os tipos dos argumentos */
    if (ARGaux->ENUM_OU_ESTR == ENU)
      fprintf(CLIENTE,"enum %s ",ARGaux->TIPO);
     else
      if (ARGaux->ENUM_OU_ESTR == EST)
        fprintf(CLIENTE,"struct %s ",ARGaux->TIPO);
       else
        if (strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
          fprintf(CLIENTE,"%s ",ARGaux->TIPO);
         else
          fprintf(CLIENTE,"char ");
    /* escreve os nomes dos argumentos */
    if (! strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
      if (ARGaux->PRIM_IND != NULL)
        fprintf(CLIENTE,"%s[%s][LEN_MAX_STR]",ARGaux->VARIAVEL,
              ARGaux->PRIM_IND->IND);
       else
        fprintf(CLIENTE,"%s[LEN_MAX_STR]",ARGaux->VARIAVEL,
              ARGaux->PRIM_IND->IND);
     else
      if (ARGaux->PRIM_IND != NULL)
        fprintf(CLIENTE,"%s[%s]",ARGaux->VARIAVEL,
              ARGaux->PRIM_IND->IND);
       else
        if (ARGaux->PARAMETRO == IN)
          fprintf(CLIENTE,"%s",ARGaux->VARIAVEL);
         else
          fprintf(CLIENTE,"*%s",ARGaux->VARIAVEL);
    ARGaux = ARGaux->PROXIMO;
    if (ARGaux != NULL)
      fprintf(CLIENTE,",");
```

```
        }
      /* escreve as declaracoes das variaveis utilizadas */
      fprintf(CLIENTE,")\n");
      fprintf(CLIENTE,"  {\n");
      fprintf(CLIENTE,"   int TAM_MSG = 0;\n");
      fprintf(CLIENTE,"   char *SRV = \"%s\\0\";\n",SUBaux->SERVIDOR);
      fprintf(CLIENTE,"   BYTE MENSAGEM[3000];\n");
      fprintf(CLIENTE,"   union TIPO_PRE ARGUMENTO;\n");
      fprintf(CLIENTE,"   int I_1=0, J_1=0;\n");
      /* no caso de funcao, escreve a declaracao do tipo e
        nome do valor de retorno */
      if (strcmp(SUBaux->TIPO,RESERVADAS[VOID]))
        if (SUBaux->ENUM_OU_ESTR == ENU)
          fprintf(CLIENTE,"   enum %s RETORNO;\n",SUBaux->TIPO);
        else
          if (SUBaux->ENUM_OU_ESTR == EST)
            fprintf(CLIENTE,"   struct %s RETORNO;\n",SUBaux->TIPO);
          else
            if (strcmp(SUBaux->TIPO,RESERVADAS[STRING]))
              fprintf(CLIENTE,"   %s RETORNO;\n",SUBaux->TIPO);
            else
              fprintf(CLIENTE,"   char *RETORNO;\n");
      /* aloca espaco para a variavel union ARGUMENTO */
/*      fprintf(CLIENTE,"   ARGUMENTO = (union TIPO_PRE *)malloc"); *
      fprintf(CLIENTE,"(sizeof(union TIPO_PRE));\n");    */
      /* escreve as funcoes invariantes */
      fprintf(CLIENTE,"   Inicializa();\n");
      /* escreve os procedimentos de montar mensagem */
      /* com os argumentos de entrada */
      PARaux = PROCaux->ENT;
      EscreveArgToMsg(CLIENTE,PARaux,IN);
      /* com os argumentos de entrada e saida */
      PARaux = PROCaux->ENT_SAI;
      EscreveArgToMsg(CLIENTE,PARaux,INOUT);
      /* escreve a chamada de procedimento remoto */
      fprintf(CLIENTE,
            "   ExecutarProcedimento(SRV,%s,PROC_%02d,",
           SUBaux->PROTOCOLO,CONTADOR);
      fprintf(CLIENTE,"MENSAGEM,&TAM_MSG);\n");
      /* escreve as funcoes invariantes */
      /* escreve os procedimentos de desmontar mensagem */
      /* com os argumentos de return */
      PARaux = PROCaux->RET;
      EscreveMsgToArg(CLIENTE,PARaux,0);
      /* com os argumentos de entrada e saida */
      PARaux = PROCaux->ENT_SAI;
      EscreveMsgToArg(CLIENTE,PARaux,INOUT);
      /* com os argumentos de saida */
      PARaux = PROCaux->SAI;
      EscreveMsgToArg(CLIENTE,PARaux,OUT);
/*    fprintf(CLIENTE,"   free(ARGUMENTO);\n"); */
      /* no caso de funcao, escreve o comando  "return" */
      if (strcmp(SUBaux->TIPO,RESERVADAS[VOID]))
        fprintf(CLIENTE,"   return(%s);\n",PROCaux->RET->NOME_TP);
      fprintf(CLIENTE,"  }\n\n");
      PROCaux = PROCaux->PROXIMO;
```

```
      SUBaux = SUBaux->PROXIMO;
     }
   }


/*    fprintf(SERVIDOR,"    ARGUMENTO = (union TIPO_PRE *)malloc");
      fprintf(SERVIDOR,"(sizeof(TAM_TIPO_PRE));\n"); */
 void CriaServerStub()
  {
   int CONTADOR = 0;
   struct SUBROTINA *SUBaux;
   struct ARGUMENTO *ARGaux;
   struct INDICE *INDaux;
   struct ELEMENTO_SUB *PROCaux;
   struct PARAMETRO *PARaux;
   enum OPCAO EXISTE_INDICE;
   SUBaux = LISTA_SUBROTINA;
   PROCaux = TABELA_SUB;
   while (SUBaux != NULL)
    {
     CONTADOR++;
     /* escreve a definicao da funcao */
     fprintf(SERVIDOR,
          " int Procedimento_%02d(int *TAM_MSGent, BYTE MENSAGEM[])\n",
          CONTADOR);
     fprintf(SERVIDOR,"  {\n");
     /* escreve as declaracoes das variaveis utilizadas */
     fprintf(SERVIDOR,"    int TAM_MSG = 0;\n");
     fprintf(SERVIDOR,"    union TIPO_PRE ARGUMENTO;\n");
     /* no caso de funcao, escreve a declaracao do tipo e
       nome do valor de retorno */
     if (strcmp(SUBaux->TIPO,RESERVADAS[VOID]))
       if (SUBaux->ENUM_OU_ESTR == EST)
         fprintf(SERVIDOR,"    struct %s RETORNO;\n",
              SUBaux->TIPO);
        else
         if (SUBaux->ENUM_OU_ESTR == ENU)
           fprintf(SERVIDOR,"    enum %s RETORNO;\n",
                SUBaux->TIPO);
          else
           if (! strcmp(SUBaux->TIPO,RESERVADAS[STRING]))
             fprintf(SERVIDOR,"    char RETORNO[LEN_MAX_STR];\n");
            else
             fprintf(SERVIDOR,"    %s RETORNO;\n",
                  SUBaux->TIPO);
     /* escreve a declaracao do tipo e nome dos argumentos */
     /* dos valores de entrada */
     ARGaux = SUBaux->PRIM_ARG;
     fprintf(SERVIDOR,"    /* argumentos de entrada */\n");
     while (ARGaux != NULL)
      {
       /* escreve os tipos dos argumentos */
       if (ARGaux->ENUM_OU_ESTR == ENU)
         fprintf(SERVIDOR,"    enum %s ",ARGaux->TIPO);
        else
         if (ARGaux->ENUM_OU_ESTR == EST)
```

```
        fprintf(SERVIDOR,"    struct %s ",ARGaux->TIPO);
       else
        if (strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
          fprintf(SERVIDOR,"    %s ",ARGaux->TIPO);
         else
          fprintf(SERVIDOR,"    char ");
    /* escreve os nomes dos argumentos */
    if (! strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
      if (ARGaux->PRIM_IND != NULL)
        fprintf(SERVIDOR,"%s[%s][LEN_MAX_STR];\n",ARGaux->VARIAVEL,
              ARGaux->PRIM_IND->IND);
       else
        fprintf(SERVIDOR,"%s[LEN_MAX_STR];\n",ARGaux->VARIAVEL);
     else
      if (ARGaux->PRIM_IND != NULL)
        fprintf(SERVIDOR,"%s[%s];\n",ARGaux->VARIAVEL,
              ARGaux->PRIM_IND->IND);
       else
        if (ARGaux->PARAMETRO == IN)
          fprintf(SERVIDOR,"%s;\n",ARGaux->VARIAVEL);
         else
          fprintf(SERVIDOR,"*%s;\n",ARGaux->VARIAVEL);
    ARGaux = ARGaux->PROXIMO;
   }
  fprintf(SERVIDOR,"    int I_1=0, J_1=0;\n");
  fprintf(SERVIDOR,"    TAM_MSG = *TAM_MSGent;\n");
  /* imprime o malloc para os ponteiros */
  ARGaux = SUBaux->PRIM_ARG;
  fprintf(SERVIDOR,"    /* malloc */\n");
  while (ARGaux != NULL)
   {
    if ((((ARGaux->PARAMETRO == OUT) ||
      (ARGaux->PARAMETRO == INOUT))) &&
      (ARGaux->PRIM_IND == NULL))
     {
      if (ARGaux->ENUM_OU_ESTR == ENU)
       {
        fprintf(SERVIDOR,
         "    %s = (enum %s *)malloc",
         ARGaux->VARIAVEL,ARGaux->TIPO);
        fprintf(SERVIDOR,"(sizeof(enum %s));\n",ARGaux->TIPO);
       }
      else
       if (ARGaux->ENUM_OU_ESTR == EST)
        {
         fprintf(SERVIDOR,
          "    %s = (struct %s *)malloc",
          ARGaux->VARIAVEL,ARGaux->TIPO);
         fprintf(SERVIDOR,"(sizeof(struct %s));\n",ARGaux->TIPO);
        }
       else
        if (strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
         {
          fprintf(SERVIDOR,
           "    %s = (%s *)malloc",
           ARGaux->VARIAVEL,ARGaux->TIPO);
```

```
                fprintf(SERVIDOR,"(sizeof(%s));\n",ARGaux->TIPO);
               }
            }
           ARGaux = ARGaux->PROXIMO;
          }
         /* aloca espaco para a variavel union ARGUMENTO */
/*       fprintf(SERVIDOR,"    ARGUMENTO = (union TIPO_PRE *)malloc");
         fprintf(SERVIDOR,"(sizeof(TAM_TIPO_PRE));\n"); */
         /* escreve os procedimentos de desmontar mensagem */
         /* com os argumentos de entrada */
         PARaux = PROCaux->ENT;
         EscreveMsgToArg(SERVIDOR,PARaux,IN);
         /* com os argumentos de entrada e saida */
         PARaux = PROCaux->ENT_SAI;
         EscreveMsgToArg(SERVIDOR,PARaux,INOUT);
         /* escreve a chamada da funcao/procedimento */
         /* no caso de funcao, escreve recebendo o valor de retorno */
         if (strcmp(SUBaux->TIPO,RESERVADAS[VOID]))
           fprintf(SERVIDOR,"    RETORNO = %s(",SUBaux->NOME);
          else
           fprintf(SERVIDOR,"    %s(",SUBaux->NOME);
         /* escreve os argumentos de passagem */
         ARGaux = SUBaux->PRIM_ARG;
         while (ARGaux != NULL)
          {
           if (ARGaux->ENUM_OU_ESTR == ENU)
             if (ARGaux->PARAMETRO == IN)
               fprintf(SERVIDOR,"%s",ARGaux->VARIAVEL);
              else
               fprintf(SERVIDOR,"%s",ARGaux->VARIAVEL);
            else
             if (ARGaux->ENUM_OU_ESTR == EST)
               if (ARGaux->PARAMETRO == IN)
                 fprintf(SERVIDOR,"%s",ARGaux->VARIAVEL);
                else
                 fprintf(SERVIDOR,"%s",ARGaux->VARIAVEL);
               else
                if (! strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
                  fprintf(SERVIDOR,"%s",ARGaux->VARIAVEL);
                 else
                  if (ARGaux->PARAMETRO == IN)
                    fprintf(SERVIDOR,"%s",ARGaux->VARIAVEL);
                   else
                    fprintf(SERVIDOR,"%s",ARGaux->VARIAVEL);
            ARGaux = ARGaux->PROXIMO;
            if (ARGaux != NULL)
              fprintf(SERVIDOR,",");
          }
         fprintf(SERVIDOR,");\n");
/*       fprintf(SERVIDOR,"    MENSAGEM[0] = (BYTE) '\\0';\n"); */
         /* escreve os procedimentos de montar mensagem */
         /* com os argumentos de return */
         PARaux = PROCaux->RET;
         EscreveArgToMsg(SERVIDOR,PARaux,0);
         /* com os argumentos de entrada e saida */
         PARaux = PROCaux->ENT_SAI;
```

```
      EscreveArgToMsg(SERVIDOR,PARaux,INOUT);
      /* com os argumentos de saida */
      PARaux = PROCaux->SAI;
      EscreveArgToMsg(SERVIDOR,PARaux,OUT);
      fprintf(SERVIDOR,"    *TAM_MSGent = TAM_MSG;\n");
      /* copia os argumentos de saida para o buffer */
/*    fprintf(SERVIDOR,"    free(ARGUMENTO);\n");     */
      /* imprime o free para os ponteiros */
      ARGaux = SUBaux->PRIM_ARG;
      fprintf(SERVIDOR,"    /* free */\n");
      while (ARGaux != NULL)
       {
        if ((((ARGaux->PARAMETRO == OUT) ||
          (ARGaux->PARAMETRO == INOUT))) &&
          (ARGaux->PRIM_IND == NULL))
          if (strcmp(ARGaux->TIPO,RESERVADAS[STRING]))
            fprintf(SERVIDOR,
             "    free(%s);\n",ARGaux->VARIAVEL);
        ARGaux = ARGaux->PROXIMO;
       }
      fprintf(SERVIDOR,"    return(0);\n");
      fprintf(SERVIDOR,"   }\n\n");
      PROCaux = PROCaux->PROXIMO;
      SUBaux = SUBaux->PROXIMO;
     }
   fprintf(SERVIDOR," void IniciaVariaveis()\n");
   fprintf(SERVIDOR,"   {\n");
/* fprintf(SERVIDOR,"    int I;\n");
   fprintf(SERVIDOR,"    BYTE\n");
   for (CONTADOR=1; CONTADOR<=NUM_PROCS-1; CONTADOR++)
     fprintf(SERVIDOR,"      resp_%02d[DMaxSes][DLenPacote],\n",CONTADOR);
   fprintf(SERVIDOR,"      resp_%02d[DMaxSes][DLenPacote];\n",NUM_PROCS);
   fprintf(SERVIDOR,"    for (I=0; I<DMaxSes; I++)\n");
   fprintf(SERVIDOR,"      {\n");
   for (CONTADOR=1; CONTADOR<=NUM_PROCS; CONTADOR++)
     fprintf(SERVIDOR,"       resp_%02d[I][0] = '\\0';\n",
          CONTADOR);
   for (CONTADOR=1; CONTADOR<=NUM_PROCS; CONTADOR++)
     fprintf(SERVIDOR,"       AtribueResp(I,PROC_%02d,resp_%02d[I]);\n",
          CONTADOR,CONTADOR);
   for (CONTADOR=1; CONTADOR<=NUM_PROCS; CONTADOR++)
     fprintf(SERVIDOR,"       TAB_SES[I].RESP[PROC_%02d] = resp_%02d[I];\n",
          CONTADOR,CONTADOR);
   fprintf(SERVIDOR,"      }\n");   */
   for (CONTADOR=1; CONTADOR<=NUM_PROCS; CONTADOR++)
     fprintf(SERVIDOR,"    AtribuePrimitiva(PROC_%02d,Procedimento_%02d);\n",
          CONTADOR,CONTADOR);
/* for (CONTADOR=1; CONTADOR<=NUM_PROCS; CONTADOR++)
     fprintf(SERVIDOR,"    Primitiva[PROC_%02d] = Procedimento_%02d;\n",
          CONTADOR,CONTADOR); */
   fprintf(SERVIDOR,"   }\n\n");
   fprintf(SERVIDOR," void main()\n");
   fprintf(SERVIDOR,"   {\n");
   fprintf(SERVIDOR,"    IniciaVariaveis();\n");
   fprintf(SERVIDOR,"    Servidor();\n");
   fprintf(SERVIDOR,"   }\n");
```

```
}

void Gerador()
  {
   printf("Gerador:\n");
   CriaHeader();
   CriaClientStub();
   CriaServerStub();
  }
```