UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

"IMPLEMENTAÇÃO DO ELEMENTO DE SERVIÇO PARA OPERAÇÕES REMOTAS UTILIZANDO UMA FERRAMENTA DE CASE CONHECIDA POR EPOS"

AUTOR: CLÁUDIO RODRIGUES MUNIZ DA SILVA
ORIENTADOR: PROF. DR. MANUEL DE JESUS MENDES

Tese apresentada à Faculdade de Engenharia Elétrica - FEE - UNICAMP, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

Junho de 1991

Este exemplar corresponde à redação final da te
defendida por Cláudio Rodrigues Muniz da Silva
e aprovada pela Comiss
julgadora em 03 / junho / 91.

Orientador

Aos Meus Pais,
à Iracema e
à minha familia.

A B S T R A C T

This work shows an implementation specification example of an Application layer service element of the OSI Model. It is a protocol that supports several other service elements of the same layer in the remote operations invocation. It is known as ROSE (Remote Operations Service Element).

This protocol is implemented using a CASE tool, known as EPOS, with two objectives: analyze the use of EPOS in the implementation of protocols and integrate the code generated to the SISDI_MAP system of the Computer Engineering Department of UNICAMP.

S U M Á R I O

Este trabalho mostra um exemplo de especificação de implementação para um Elemento de Serviço da Camada de Aplicação do modelo OSI. Trata-se de um protocolo que dá suporte à invocação de operações por outros Elementos de Serviço da mesma camada, e é conhecido por ROSE (Remote Operations Service Element).

A Implementação deste protocolo é feita através do uso da ferramenta CASE conhecida por EPOS e tem como objetivos principais, analisar a adequação do EPOS para a área de desenvolvimento de protocolos e integrar o código final gerado para o ROSE no sistema SISDI_MAP do Departamento de Engenharia da Computação e Automação Industrial da UNICAMP.

RELAÇÃO DE FIGURAS

Figura	Descrição
1.1	Esquema Genérico do SISDI_MAP e suas Extensões
1.2	Estrutura de Desenvolvimento do EPOS
1.3	Modelo de Projeto do EPOS
2.1	Contexto de Aplicação com Elemento RTSE
2.2	Contexto de Aplicação sem Elemento RTSE
2.3	Esquema de Mapeamento Geral em Contextos sem RTSE
2.4	Esquema de Mapeamento Geral em Contextos com RTSE
3.1	Modelo de Implementação para uma Camada N
3.2	Modelo para Implementação do ROSE
3.3	Modelo Geral de uma Especificação em EPOS-S
3.4	Esquema Hierárquico dos Módulos principais da Especificação em EPOS-S
3.5	Diagrama DATAFLOW da Ação PROC_ROSE com RTSE
3.6	Diagrama DATAFLOW da Ação PROC_ROSE sem RTSE
3.7	Diagrama DATAFLOW da Ação TREAT_SASE
3.8	Diagrama DATAFLOW da Ação TREAT_RTSE
3.9	Diagrama DATAFLOW da Ação TREAT_RO_INV_REQ
3.10	Diagrama DATAFLOW da Ação EXEC_TRANS_REQ
3.11	Diagrama DATAFLOW da Ação TREAT_RT_TR_IND
3.12	Diagrama DATAFLOW da Ação TREAT_RT_TR_CNF_POS
3.13	Diagrama DATAFLOW da Ação TREAT_RT_TR_CNF_NEG
3.14	Diagrama DATAFLOW da Ação TREAT_RT_TP_IND
3.15	Diagrama DATAFLOW da Ação TREAT_RT_TG_IND
3.16	Diagrama DATAFLOW da Ação TREAT_ABORT_IND
3.17	Diagrama DATAFLOW da Ação TREAT_P_DATA_IND
3.18	Diagrama DATAFLOW da Ação TREAT_AA_ESTAB
3.19	Diagrama DATAFLOW da Ação TREAT_AA_REL
3.20	Esquema de Comunicação Externa do ROSE
3.21	Esquema de Sinalização de Estabelecimento/Liberação de Associação ao ROSE
4.1	Bloco de Mensagem para Primitiva ro_invoke.req
4.2	Bloco de Mensagem para Primitiva p_data_req
4.3	Bloco de Mensagem para Primitiva p_data_ind

Relação de Tabelas

Tabela	Descrição
2.1	Relação de Serviços do ROSE
2.2	Campos da APDU ROIV
2.3	Campos da APDU RORS
2.4	Campos da APDU ROER
2.5	Campos da APDU RORJu
2.6	Campos da APDU RORJp
2.7	Relação de Estados da ROPM
2.8	Relação de Estados da ROPM-TR
2.9	Relação de Eventos de Entrada
2.10	Relação de Ações da Máquina de Estados do ROSE
2.11	Relação de Predicados (condições) usados
2.12	Tabela de Eventos/Estados da ROPM
2.13	Tabela de Eventos/Estados da ROPM-TR
2.14	Tabela de Eventos/Estados da ROPM-TR (AE sem RTSE)
3.1	Exemplo da Tabela de Controle de Conexões e Estados
3.2	Tabela de Primitivas que originam os Eventos AA_ESTAB e AA_REL
3.3	Análise Comparativa entre Percentuais de Codificação Automática/Manual na Implementação que Envolve o CASE RTSE no contexto de Aplicação;
3.4	Análise Comparativa entre Percentuais de Codificação Automática/Manual na Implementação que não Envolve o CASE RTSE no contexto de Aplicação;

ÍNDICE

1. Introdução 1-1

1.1. O Projeto SISDI_MAP 1-2
1.2. O Software EPOS 1-5

2. O Elemento de Serviço para Operações Remotas - ROSE 2-1

2.1. As Operações Remotas 2-3
2.2. A Notação das Operações Remotas - "RO-NOTATION" 2-6
2.3. A Especificação dos Serviços do ROSE 2-16
2.4. A Especificação do Protocolo do ROSE 2-22

2.4.1. Elementos de Procedimento 2-22
2.4.2. Máquina de Estados do ROSE 2-27

3. A Especificação da Implementação 3-1

3.1. Especificação dos Requisitos Utilizando o EPOS-R 3-6
3.2. Especificação formal da implementação em EPOS-S 3-10

3.2.1. Os Objetos de Projeto 3-11
3.2.2. Filosofias de Especificação para Implementação .. 3-12
3.2.3. A Implementação 3-14

3.3. Geração Automática de Código 3-43

4. Exemplo de Funcionamento do Processo ROSE 4-1

5. Conclusões 5-1

Referências Bibliográficas 6-1

Apêndice A - Arquivos de Includes referentes ao ROSE A-1

Apêndice B - Análises Feitas pelo EPOS - S B-1

1. INTRODUÇÃO

A corrida do mundo moderno aos representantes da era da informação, os computadores, aumenta a cada dia a presença e importância da informação na vida dos homens. A princípio, sua manipulação era localizada, presa e limitada aos recursos de um único equipamento; depois, com a necessidade de compartilhamento dos recursos computacionais e o desenvolvimento de novas tecnolo-gias na eletrônica e micro-eletrônica, surgiram as chamadas Redes de computadores. A informação, então, adquiriu uma mobilidade surpreendente e, como consequência, teve sua importância aumentada em muito.

No início, as redes existentes interligavam apenas equipamentos de um mesmo fabricante, como foi o caso da IBM e da DEC com suas redes SNA e DNA, respectivamente. Ficava, no entanto, o problema de como interligar equipamentos de fabricantes diferentes. Como fazer computadores com características elétricas, programas e formato de dados diferentes se comunicarem sem problemas ?

Algumas propostas surgiram, tal como criar conversores específicos entre os sistemas existentes. Porém, eram soluções lentas e ineficientes para manipular adequadamente os diversos dados que uma aplicação deve tratar.

Em 1977, a ISO, Organização Internacional de Padrões, criou o comitê SC16 com o objetivo de estudar a problemática e definir padrões de comunicação. Em 1983, como resultado dos estudos feitos, a ISO aprovou um conjunto de convenções, arquitetura, para a interconexão de sistemas. Os sistemas que adotassem tal arquitetura passariam a chamar-se Sistemas Abertos, em inglês, "OSI systems".

O modelo apresentado como a arquitetura dos sistemas abertos é formado por sete camadas dispostas hierárquicamente e com funções bem definidas.

Um subconjunto do modelo OSI compõe a arquitetura MAP, "Manufacturing Automation Protocol" [1]; sua função é padronizar a comunicação no ambiente das manufaturas. A arquitetura MAP foi a base para a criação do projeto SISDI_MAP no DCA-UNICAMP [2]. O projeto inicial previa a implementação parcial do MAP. Algumas camadas seriam implementadas inteiramente, outras parcialmente e uma se encarregaria de simular as restantes.

A fase inicial foi concluída com a implementação manual de todos os seu módulos componentes. Algumas extensões ao projeto inicial também foram planejadas. Entre elas, a inclusão das camadas de Sessão e Apresentação e, também, porém mais tarde, a introdução do protocolo ROSE [3] e seus ASE's relacionados: RTSE [4] e um "ROSE-USER".

Este trabalho apresenta a especificação de implementação do protocolo ROSE feita através de uma ferramenta de engenharia de "software" conhecida por EPOS [5].

A apresentação descreve nas seções seguintes deste capítulo, o projeto SISDI_MAP e o ambiente EPOS, seções 1.1. e 1.2, respectivamente. No capítulo 2, detalha o protocolo ROSE e a notação especial que acompanha seus documentos de padronização. O capítulo 3, descreve todos os aspectos do desenvolvimento de uma especificação de implementação para o ROSE e, por fim, no capítulo 4 são mostrados vários exemplos de funcionamento do programa final criado.

1.1. O Projeto SISDI_MAP

O SISDI_MAP, Sistema Didático para MAP [2], é um projeto em desenvolvimento no departamento de Engenharia da Computação e Auto-mação Industrial da FEE-UNICAMP. Seu objetivo é simular a comunicação entre dois nós ligados em Rede e em um ambiente industrial, monitorando sequências de dados transmitidos e inserindo, se pedido, erros nas PDU's das camadas envolvidas. Os protocolos de comunicação usados são os previstos na arquitetura MAP,

Manufacturing Automation Protocol.

O SISDI_MAP é formado por um "software" básico e um conjunto de tarefas que formam um "software" aplicativo. O "software" básico é constituido por um núcleo de tempo real dedicado que gerencia a comunicação entre as tarefas que correspondem às camadas definidas na arquitetura MAP.

A definição inicial do projeto incluia as seguintes tarefas:

- Interface de Operação do Usuário;
- Interface de Aplicação (API);
- Protocolo MMS;
- Protocolo ACSE;
- Simulador dos Serviços da Apresentação.

Atualmente, o projeto foi estendido no sentido de incluir duas novas camadas: Sessão [6] e Apresentação [7], ficando as funções de simulação contidas agora em um processo simulador da camada de Transporte [8].

A idéia de incluir um protocolo que não pertence à versão do MAP disponível, como o ROSE, tem como objetivo prever uma possível evolução da especificação atual para uma em que o protocolo seja especificado através da notação das operações remotas. Outra razão seria o fato de que, uma vez implementado o ROSE, poderia se criar um novo sistema didático voltado para uma outra aplicação diferente de MAP. Uma opção já em desenvolvimento é o protocolo de acesso a banco de dados distribuidos conhecido como RDA [9].

A extensão atual planejada para o SISDI_MAP inclui os seguintes protocolos: ROSE, RTSE e RDA, conforme a Figura 1.1.

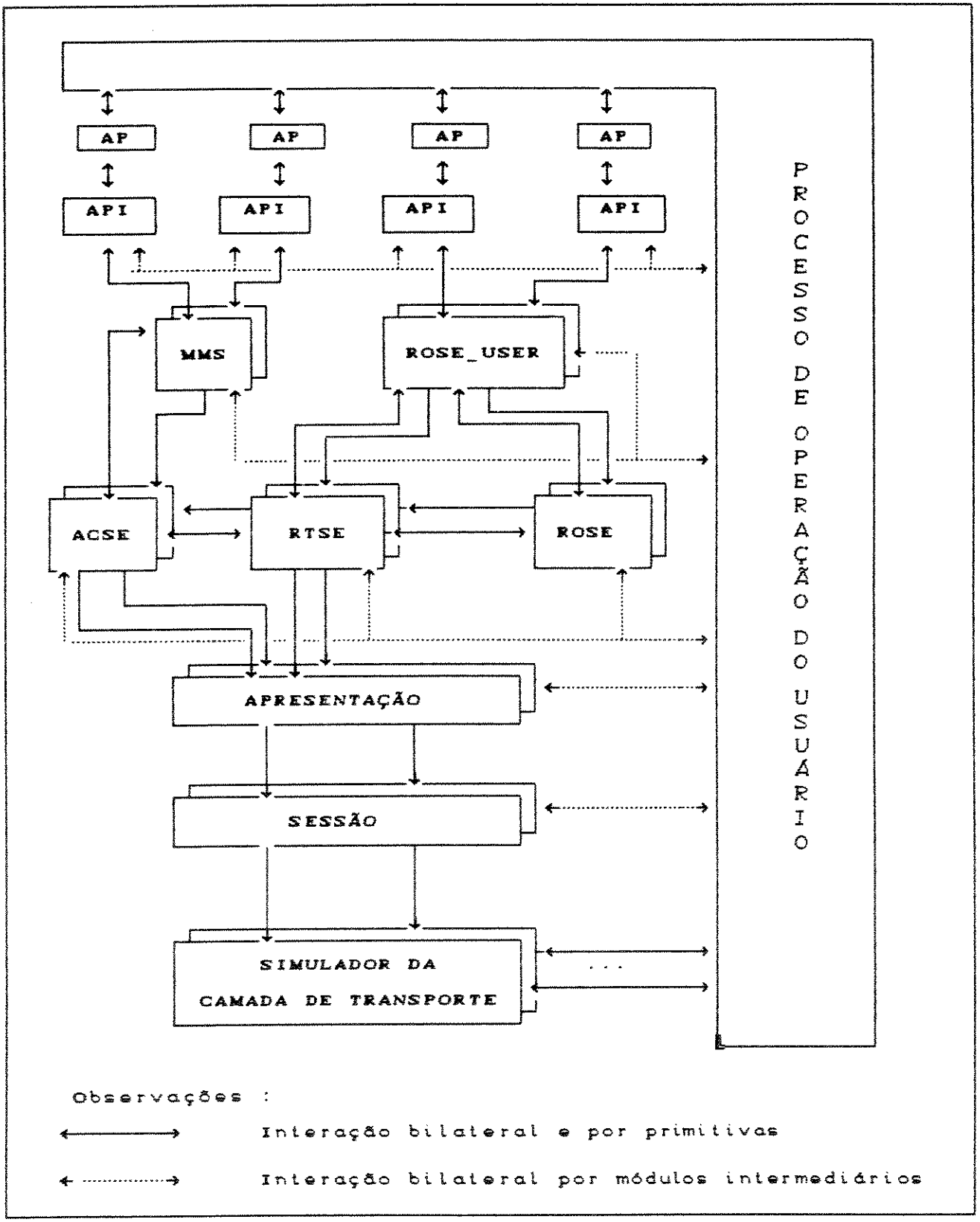


Figura 1.1 - Esquema Genérico do SISDI_MAP e suas Extensões

1.2. O SOFTWARE EPOS

O pacote EPOS [11], "Engineering Project-Oriented Specification", é um pacote criado para dar suporte ao projeto e desenvolvimento de sistemas de "software" e "hardware". O suporte oferecido consiste em:

- especificar os projetos de forma fácil e independente de uma linguagem específica;
- documentar todos os passos de um projeto durante o desenvolvimento do mesmo e não depois de terminado o mesmo;
- auxiliar no planejamento de projetos desenvolvidos separadamente, porém interligados funcionalmente.

O EPOS compõe-se basicamente de um conjunto de módulos com funções bem definidas que interagem entre si para formar um ambiente de desenvolvimento de projetos. Os módulos são:

EPOS-R - É uma linguagem de especificação dos requisitos, restrições e modelo matemático de implementação de um sistema. Possui um grau ajustável de formalidade.

EPOS-S - É uma linguagem de especificação formal com sintaxe e semântica rígidas, usada para descrever o projeto de um sistema.

EPOS-P - Trata-se, também, de uma linguagem de especificação. Descreve informações importantes para o gerenciamento do projeto e configuração de gerenciamento.

EPOS-C - Este módulo compreende todo o sistema de comunicação com o usuário. Dá suporte a entrada de especificações, gerenciamento do banco de dados, análise e geração de relatórios pelo EPOS.

EPOS-A - É uma ferramenta de análise de erros nas três especificações feitas anteriormente, gerando para isso relatórios da situação atual do banco de dados.

EPOS-D - É o módulo responsável pela geração da documentação textual e gráfica das informações encontradas no banco de dados.

EPOS-M - Dá suporte às atividades relacionadas ao projeto e configuração de gerenciamento.

COMPOSER - É a ferramenta de geração automática de código. Sua função é passar a especificação de implementação feita no EPOS-S para uma linguagem de programação específica.

Ferramenta de Apoio aos Métodos de Projeto - Dá suporte no projeto de um sistema que use um dos métodos adotados pelo EPOS.

A figura a seguir mostra um esquema geral do EPOS:

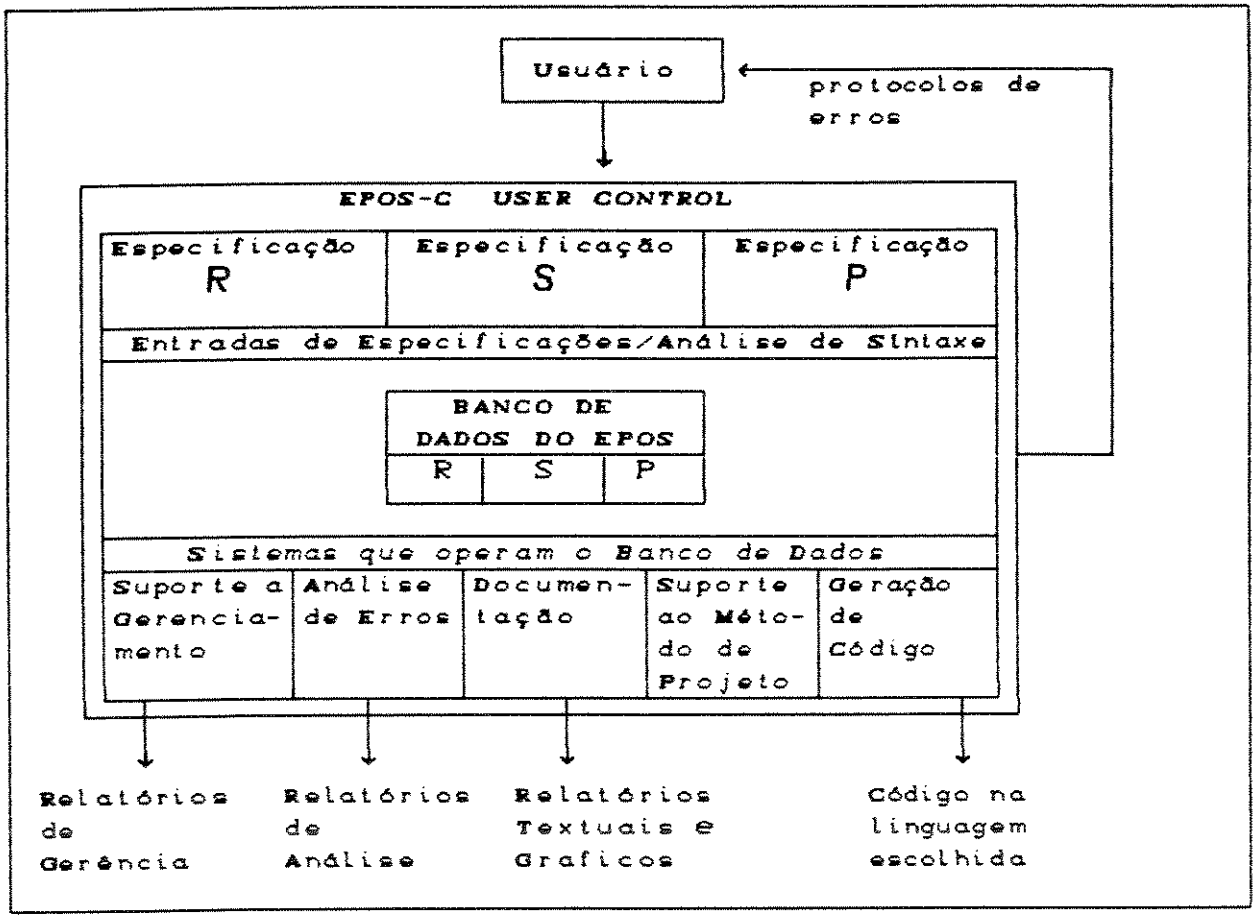


Figura 1.2 - Estrutura de Desenvolvimento do EPOS

Na Figura 1.2, os relatórios de gerência são fornecidos pelo EPOS-M, os relatórios de análise pelo EPOS-A, os relatórios textuais e gráficos em plotter/monitor pelo EPOS-D e os códigos pelo módulo Composer.

A Figura 1.3 mostra, em esquema gráfico, as etapas que compõem o desenvolvimento de uma aplicação EPOS e a interação existente entre elas. Cada uma das etapas é subdividida em outros módulos sobre os quais se mostram também as interrelações.

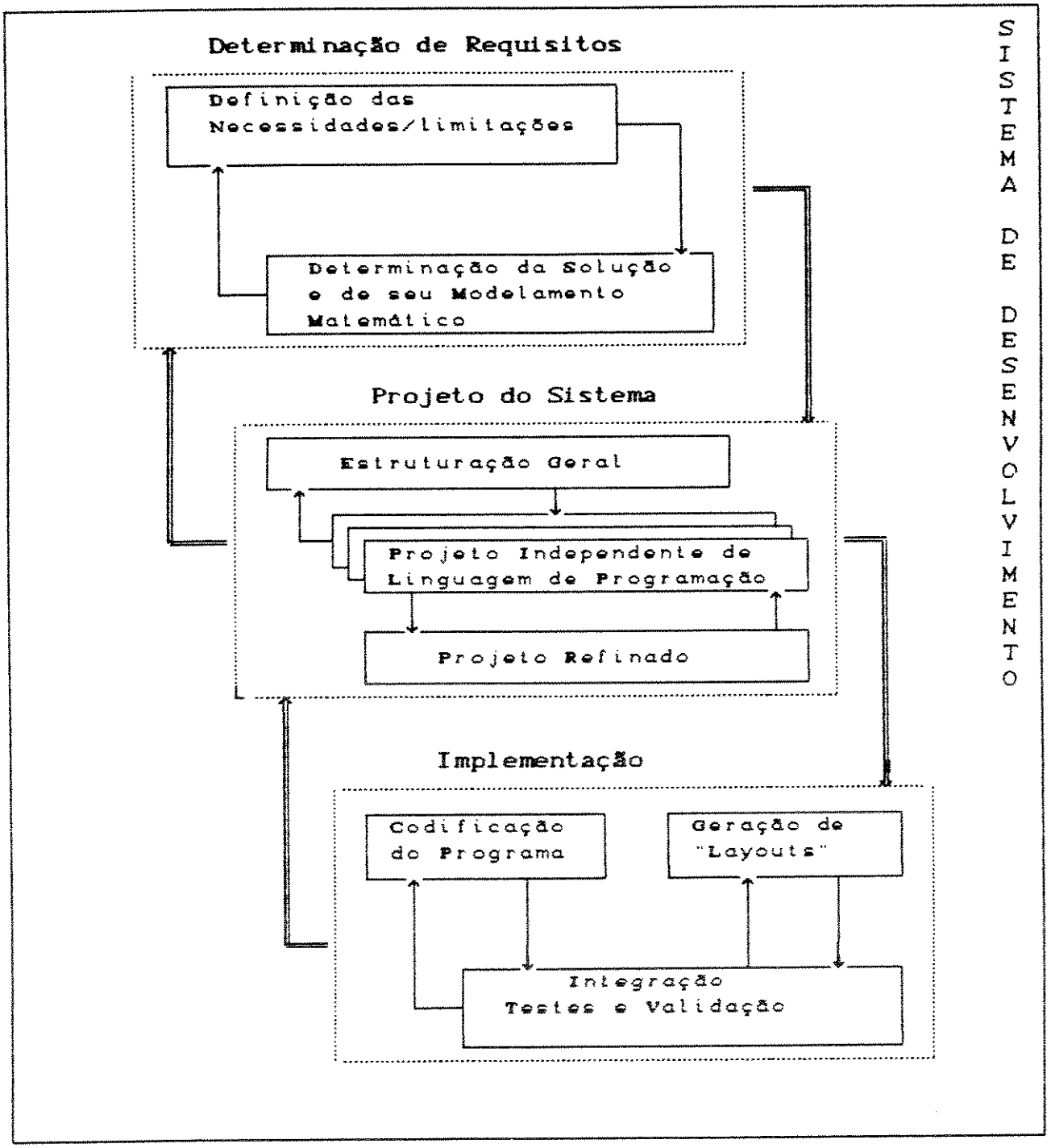


Figura 1.3 - Modelo de Projeto do EPOS

2. O Elemento de Serviço para Operações Remotas - ROSE

No ambiente OSI, a comunicação entre Processos de Aplicação (AP's) é modelada pela cooperação entre entidades lógicas chamadas de Entidades de Aplicação (AE's) [10]. Tais entidades englobam o comportamento observável de um AP nas suas comunicações com outros AP's e têm uma estrutura interna que reflete, tanto a comunicação em si, quanto os requisitos resultantes do seu papel no processamento da informação.

Uma Entidade de Aplicação representa um conjunto de recursos de comunicação que são definidos por um grupo de elementos de serviço conhecidos como ASE's [10]. Tais elementos são funções integradas que se encarregam do trabalho conjunto entre as Entidades de Aplicação, podendo um dado ASE ser usado independentemente ou em combinação com outros ASE's para a obtenção de uma meta específica.

Depois que duas Entidades de Aplicação que desejam se comunicar determinam que Elementos de Serviço serão usados nesta comunicação (Contexto de Aplicação), uma Associação de Aplicação é estabelecida. Apartir deste momento, os ASE's pertencentes ao Contexto de Aplicação podem fazer uso direto dos serviços da camada de Apresentação ou dos serviços de outros ASE's do mesmo contexto para que seja fornecida a funcionalidade solicitada pelo usuário dos serviços.

Pode-se classificar os ASE's em dois níveis distintos: os específicos de cada aplicação (SASE's) e os comuns a várias delas (CASE's).

Os CASE's mais conhecidos são:

- ACSE (Association Control Service Elements) [11];
- RTSE (Reliable Transfer Service Elements) [4];
- CCR (Commitment, Concurrency and Recovery) [12] e o
- ROSE (Remote Operations Service Elements) [1].

ACSE

Tem como função gerenciar conexões da camada de Aplicação (associações) entre entidades de aplicação (AE's).

É um dos mais importantes ASE's, pois toda interação entre processos de aplicação necessita primeiramente de uma conexão já estabelecida.

Cada primitiva do ACSE é mapeada sobre uma equivalente na camada de Apresentação, o que pode induzir à errônea idéia de desperdício já que se poderia usar diretamente os serviços da Apresentação. A justificativa está no fato de que o ACSE não está totalmente especificado e conseqüentemente alguns de seus serviços podem sofrer modificações que justifiquem sua existência separada. Como exemplo, teríamos o A-ASSOCIATE que pode vir a fornecer também autenticação de usuários.

RTSE

Permite uma transferência mais confiável de informação entre Entidades de Aplicação. Esta transferência mais segura é feita através de um mecanismo independente da aplicação para recuperação de erros de comunicação, minimizando o número de re-transmissões. Para tal fim, o RTSE faz uso dos serviços da camada de Rede chamados: P-MINOR-SYNCHRONIZE, P-ACTIVITY-INTERRUPT, P-ACTIVITY-DISCARD, P-ACTIVITY-RESUME, P-ACTIVITY-START e P-ACTIVITY-END e outros relacionados com controle de transmissão.

CCR

É um ASE baseado no conceito de Ação Atômica que ocorre em meio a um relacionamento do tipo Mestre/Escravo(s), onde o Processo de Aplicação Mestre solicita uma série de operações aos Processos Escravos e aguarda que todas sejam realizadas com sucesso. Se pelo menos uma não o for, as demais serão desfeitas.

ROSE

O ROSE é um protocolo que dá suporte a aplicações distribuídas do tipo "request/reply". Este ASE tem características muito especiais que levam alguns estudiosos do assunto a não classificá-lo propriamente como um simples elemento de serviço comum. A principal peculiaridade é o fato de, na sua definição, ser definido, além dos serviços e máquina de protocolo, uma técnica de projeto de Elementos Específicos chamada de Técnica das Operações Remotas. O protocolo em si é um exemplo de comunicação do tipo RPC ("Remote Procedure Call") [13].

2.1. As Operações Remotas

O modelo para as operações remotas é razoavelmente simples. Como sabe-se, na estrutura OSI, as comunicações entre Processos de Aplicação são representadas em termos das comunicações entre pares de Entidades de Aplicação usando os serviços da Apresentação. Estas comunicações, algumas vezes, são inerentemente interativas. Tipicamente, uma entidade (AE) requer que uma operação particular seja feita e aguarda a informação de status de execução da mesma. A entidade par tenta realizar o que lhe foi pedido e relata ao final se houve sucesso ou não. Chama-se interações deste tipo de interações "request/reply" ou mais precisamente de operações remotas.

As Operações Remotas podem ser classificadas quanto a seu modo de operação ou quanto ao que se espera como resposta da entidade par. Unindo as duas classificações em uma única definiu-se o conceito de Classe de Operação como sendo uma combinação específica das classificações anteriores. Existem 5 classes definidas:

Classe 1 - Síncrona, relatando sucesso ou falha;
Classe 2 - Assíncrona, relatando sucesso ou falha;
Classe 3 - Assíncrona, relatando apenas falhas;
Classe 4 - Assíncrona, relatando apenas sucesso;
Classe 5 - Assíncrona, sem nenhum relato.

Em alguns casos é util agrupar operações em conjuntos de operações relacionadas, que são conjuntos de operações formados por uma operação mãe (parent operation) e uma ou mais operações filhas (children operations). O executor da operação mãe pode invocar nenhuma, uma, ou mais operações filhas durante a sua própria execução. O invocador da operação mãe é o executor das operações filhas. Uma operação filha pode vir a ter seu próprio grupo de operações relacionadas e assim por diante.

Dependendo do tipo de associação estabelecida, três situações podem acontecer, sendo cada situação correspondente a uma classe de associação:

Classe 1 . Somente o iniciador da associação poderá invocar as operações remotas;

Classe 2 . Somente o elemento par poderá invocar as operações remotas;

Classe 3 . Ambos poderão invocar as operações remotas.

As regras que determinam que AE's invocam operações e que operações podem ser invocadas são especificadas em um módulo de operações remotas definido em ASN.1 (Abstract Syntax Notation One) [14] e que está associado ao ASE específico que utiliza o ROSE; ou seja, com o surgimento do ROSE e sua notação, os projetistas de elementos de serviço específicos não precisam mais definir uma primitiva específica para cada tarefa que precise realizar, nem mesmo definir uma máquina de protocolo que trate as mesmas, pois tudo que se queira é feito em termos de operações, usando o módulo de operações remotas e a máquina de protocolos do ROSE com seus serviços.

O Módulo de Operações Remotas é formado basicamente pelo recurso da linguagem ASN.1 conhecido por macros. O conjunto específico de macros criados para as operações remotas chama-se "RO-NOTATION". E, é este conjunto de macros, chamado também de notação das operações remotas, que torna a documentação do ROSE tão diferente dos demais elementos comuns da camada de Aplicação.

Uma importante característica das Operações Remotas é que elas dão às aplicações independência dos serviços de comunicação OSI, pois a notação é baseada em princípios estabelecidos de programação orientada a objetos que permitem o desenvolvimento de instrumentos automáticos para associar as Operações Remotas às estruturas de execução das aplicações.

O ROSE pode ser encaixado em contextos de Aplicação que utilizem o ASE RTSE ou não. Como sabe-se, o ACSE entra em todos os contextos de Aplicação. Quando tem-se um contexto que envolve RTSE e ROSE, o relacionamento entre os protocolos será:

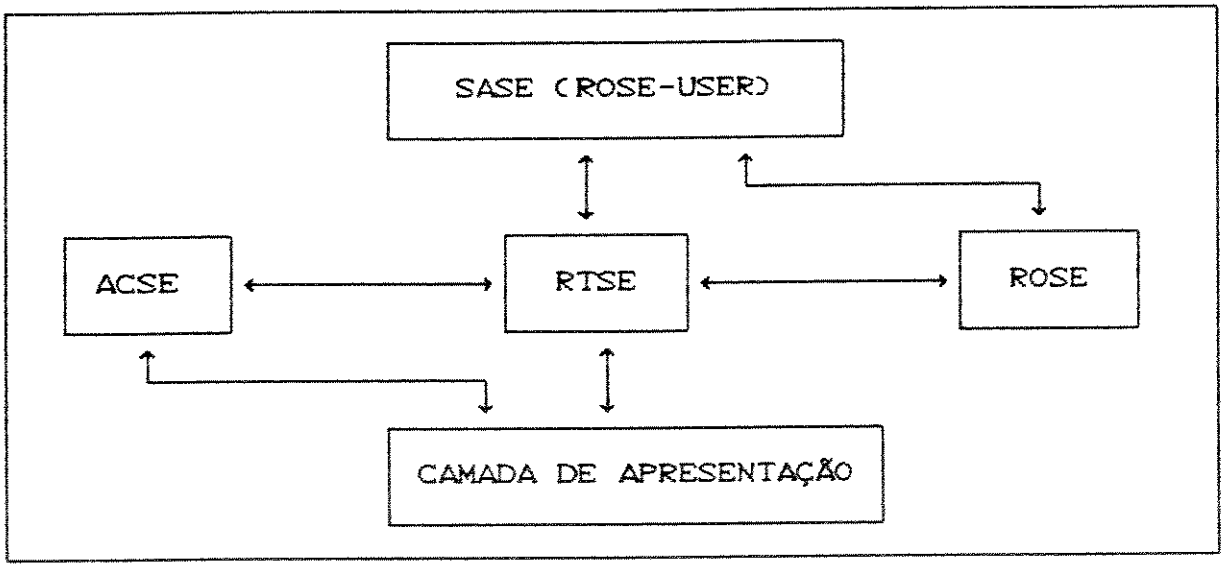


Figura 2.1 - Contexto de Aplicação com Elemento RTSE

Caso o RTSE não esteja presente no contexto, o esquema de relacionamento se modifica para o mostrado a seguir:

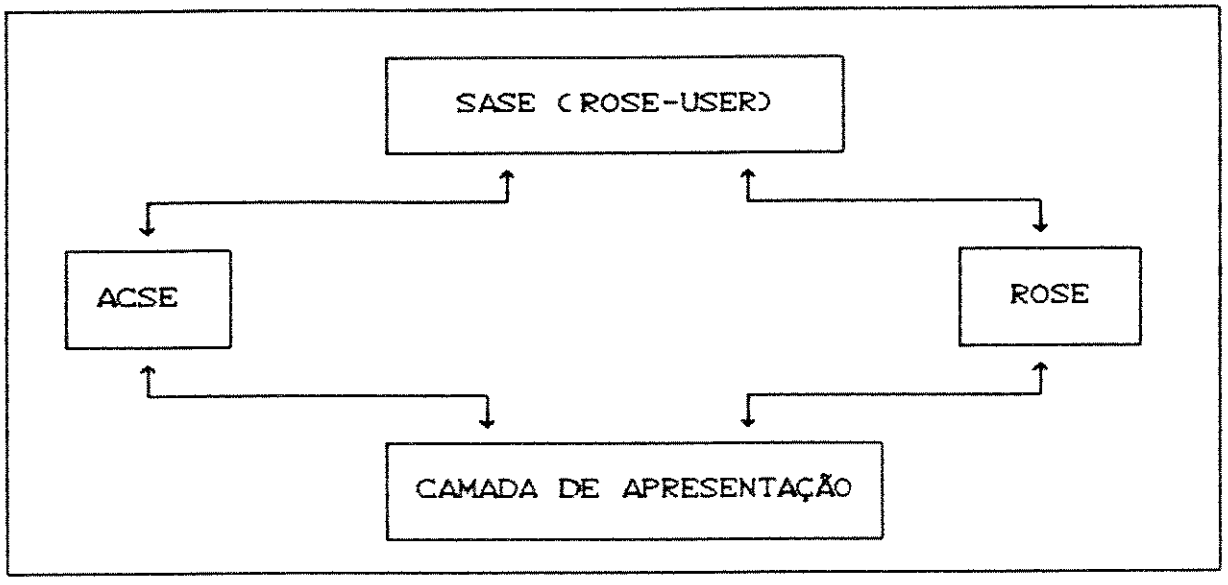


Figura 2.2 - Contexto de Aplicação sem Elemento RTSE

2.2. A Notação das Operações Remotas - "RO-NOTATION"

A notação das Operações Remotas, ou simplesmente "RO-NOTATON", é uma extensão da linguagem ASN.1, criada com o objetivo de elevar a descrição de interações complexas entre sistemas abertos, a um nível de abstração que não é alcançável por meios formais menos poderosos. A notação é usada para definir como Entidades de Aplicação OSI interagem para efetuar uma tarefa de processamento de informação específica. A notação é particularmente adequada a protocolos interativos que envolvem troca de "requests" e "replies". No entanto, pode ser aplicada a protocolos arbitrários por admitir "requests" sem resposta como caso particular.

A notação é formada por seis macros da ASN.1, que definem: contexto de aplicação, elemento de serviço, operações de estabelecimento de conexão, operações de liberação de conexões, operações genéricas e erros associados.

Macro é um recurso da ASN.1 que permite a criação de tipos de dados definidos pelo usuário, mas com uma sintaxe e semântica própria. Na criação de uma macro, utiliza-se BNF, "Backus-Nauer Form" [14], para definir-se duas produções que representam, respectivamente, uma notação de tipo ("TYPE NOTATION") e uma notação de valor ("VALUE NOTATION"). Além destas duas produções, é definida também, uma sintaxe de suporte que segue as duas notações acima.

É possível,na definição de uma macro, a introdução de tipos e valores predefinidos da linguagem ASN.1. Para tanto, faz-se uso das seguintes construções:

VALUE (variável tipo predefinido) - para referenciar um valor predefinido ou

TYPE (tipo predefinido) - para referencia um tipo predefinido.

As macros que formam a notação das operações remotas são: BIND, UNBIND, OPERATION, ERROR, APPLICATION-CONTEXT e APPLICATION-SERVICE-ELEMENT. A seguir todas as macros são definidas, através dos Módulo de Operações Remotas e sua extensão. Cada uma delas é explicada considerando-se função, notação de tipo e valor e exemplos de uso.

```
Remote-Operation {joint-iso-ccitt remote Operations(4) notation(0)}
DEFINITIONS ::=
BEGIN
EXPORT BIND, UNBIND, OPERATION, ERROR;

-- macro definition for bind-operations
BIND MACRO ::=
BEGIN
TYPE NOTATION     ::= Argument Result Error
VALUE NOTATION    ::= Argument-value | Result-value | Error-value

Argument          ::= "ARGUMENT" Name Type(Argument-type) | empty
Result            ::= "RESULT" Name Type(Result-type) | empty
Error             ::= "ERROR" Name Type(Error-type) | empty

Name              ::= identifier | type
```

```
Argument-value      ::=     "ARGUMENT" value (Arg-value Argument-type) |
                            empty
                            <VALUE[16] EXPLICIT Argument-type ::= Arg-value>
Result-value        ::=     "RESULT" value (Arg-value Result-type) | empty
                            <VALUE[17] EXPLICIT Result-type ::= Res-value>
Error-value         ::=     "ERROR" value (Arg-value Error-type) | empty
                            <VALUE[18] EXPLICIT Error-type ::= Err-value>

END

-- macro definition for unbind-operations
UNBIND MACRO ::=
BEGIN
TYPE NOTATION       ::= Argument Result Error
VALUE NOTATION      ::= Argument-value|Result-value|Error-value

Argument            ::= "ARGUMENT" Name Type(Argument-type) | empty
Result              ::= "RESULT" Name Type(Result-type) | empty
Error               ::= "ERROR" Name Type(Error-type) | empty

Name                ::= identifier | type

Argument-value      ::=     "ARGUMENT" value (Arg-value Argument-type) |
                            empty
                            <VALUE[19] EXPLICIT Argument-type ::= Arg-value>
Result-value        ::=     "RESULT" value (Arg-value Result-type) | empty
                            <VALUE[20] EXPLICIT Result-type ::= Res-value>
Error-value         ::=     "ERROR" value (Arg-value Error-type) | empty
                            <VALUE[21] EXPLICIT Error-type ::= Err-value>

END

-- macro definition for operations
OPERATION MACRO ::=
BEGIN
TYPE NOTATION             ::= Argument Result Errors LinkedOperations
VALUE NOTATION            ::= value(VALUE CHOICE
                                    < localValue INTEGER,
                                      globalValue OBJECT IDENTIFIER>)

Argument                  ::= "ARGUMENT" NamedType | empty
Result                    ::= "RESULT" ResultType | empty
ResultType                ::= NamedType | empty
Errors                    ::= "ERRORS" "{"ErrorNames"}" | empty
LinkedOperations          ::=   "LINKED" "{"LinkedOperationNames"}" |
                                empty
ErrorNames                ::= ErrorList | empty
ErrorList                 ::= Error | ErrorList "," Error
Error                     ::= value(ERROR) | type
LinkedOperationNames      ::= OperationList | empty
OperationList             ::= Operation | OperationList "," Operation
Operation                 ::= value(OPERATION) | type
NamedType                 ::= identifier type | type
END
```

```
-- macro definition for operations errors
ERROR MACRO ::=
BEGIN

TYPE NOTATION          ::= Parameter
VALUE NOTATION         ::= value(VALUE CHOICE{
                                localValue      INTEGER
                                globalValue     OBJECT IDENTIFIER})
Parameter              ::= "PARAMETER" NamedType | empty

NamedType              ::= identifier type | type
END
END


Remote-Operations-Notation-extention
{joint-iso-ccitt remote Operations(4) notation-extention(2)}

DEFINITIONS ::=
BEGIN
EXPORTS   APPLICATION-SERVICE-ELEMENT, APPLICATION-CONTEXT;
IMPORTS   OPERATION, BIND, UNBIND
          FROM Remote-Operation-Notation
               {joint-iso-ccitt remoteOperations(4) notation(0)};

-- macro definition for ASE's
APPLICATION-SERVICE-ELEMENT MACRO ::=
BEGIN

TYPE NOTATION      ::=   SymmetricASE|ConsumerInvokes SupplierInvokes|
                         empty
VALUE NOTATION     ::= value (VALUE OBJECT IDENTIFIER)

SymetricASE        ::= "OPERATIONS" "{"operationList"}"
ConsumerInvokes    ::= "CONSUMER INVOKES"{"operationList"}"|empty
SupplierInvokes    ::= "SUPPLIER INVOKES"{"operationList"}"|empty
OperationList      ::= Operation|OperationList","Operation
Operation          ::= value (OPERATION)

END

-- macro definition for application-contexts
APPLICATION-CONTEXT MACRO ::=
BEGIN

TYPE NOTATION          ::=   NonROelements     Binding     ROelements
                             AbstractSyntaxes
VALUE NOTATION         ::=   value (VALUE OBJECT IDENTIFIER)

NonROelements          ::= "APPLICATION SERVICE ELEMENTS""{"AseList"}"
Binding                ::=   "BIND"         type
                             "UNBIND"       type
ROelements             ::=   "REMOTE        OPERATIONS""{"AseID"}
                              SymmetricAses AsymmetricAses|empty
SymmetricAses          ::= "OPERATIONS OF""{"AseList"}"|empty
```

```
AsymmetricAses          ::=    InitiatorConsumerOf ResponderConsumerOf
InitiatorConsumerOf     ::= "INITIATOR CONSUMER OF""{"AseList"}" | empty
ResponderConsumerOf     ::= "RESPONDER CONSUMER OF""{"AseList"}" | empty
AbstractSyntaxes        ::= "ABSTRACT SYNTAXES""{"AbstractSyntaxList"}"
AseList                 ::=    AseID | AseList","AseID
AseID                   ::= value(APPLICATION-SERVICE-ELEMENT)
AbstractSyntaxList      ::=   AbstractSyntax | AbstractSyntaxList","
                              AbstractSyntax
AbstractSyntax          ::=   value(OBJECT IDENTIFIER)
END
END
```

BIND - Serve para definir um tipo de operação de estabelecimento de associação e seu erro associado. A notação de tipo nos diz que o tipo da macro pode ser formado por qualquer combinação dos tipo *Argument*, *Result* e *Error*. A escolha de qualquer um dos três implica na necessidade da inclusão do valor correspondente no momento de uma atribuição de valor. A macro BIND pode ser usada para definir um novo tipo de operação BIND ou criar uma instância numa variável qualquer.

- **Exemplo de Criação de Tipos de Operações BIND**

```
a) Operação-Bind-1  ::= BIND
                        ARGUMENT     Tipo-do-argumento-1
                        RESULT       Tipo-do-resultado-1
b) Operação-Bind-2  ::= BIND
                        ARGUMENT     Tipo-do-argumento-2
```

- **Exemplo de Instâncias da Macro BIND**

```
a) operação-Bind-1      BIND
                        ARGUMENT     Tipo-do-argumento-1
                        RESULT       Tipo-do-resultado-1 ::=
                        ARGUMENT     <conteúdo-do-argumento>
                        RESULT       <conteúdo-do-resultado>
b) operação-Bind-1      Operação-Bind-1    ::=
                        ARGUMENT     <conteúdo-do-argumento>
                        RESULT       <conteúdo-do-resultado>
```

UNBIND - É análoga à BIND, com a diferença que seu uso é feito para liberar uma Associação de Aplicação. Pode definir tipos, ou instâncias dos mesmos, em variáveis.

- **Exemplo de Criação de Tipos de Operações UNBIND**

```
a) Operação-Unbind-1  ::= UNBIND
                          ARGUMENT       Tipo-do-argumento-1
                          UNBIND-ERROR   Tipo-do-erro-1
b) Operação-Unbind-2  ::= UNBIND
                          RESULT     Tipo-do-resultado-2
```

- **Exemplo de Instâncias da Macro UNBIND**

```
a) operação-Unbind-1      UNBIND
                          RESULT          Tipo-do-resultado-1
                          UNBIND-ERROR    Tipo-do-erro-1  ::=
                          RESULT       <conteúdo-do-resultado>
                          ERROR        <conteúdo-do-erro>
b) operação-Bind-1      Operação-Unbind-1   ::=
                        ARGUMENT     <conteúdo-do-argumento>
                        ERROR        <conteúdo-do-erro>
```

OPERATION - É usada para representar a invocação de uma operação do conjunto permitido para a aplicação em questão. Como visto no início do capítulo, as operações podem resultar em novas invocações associadas durante sua execução. Consequentemente, além dos tipos *Argumento*, *Resultado* e *Erro*, deve existir um tipo para designar as possíveis operações "filhas" originárias da operação definida, operação "mãe".

A notação de valor muda bastante, pois uma macro OPERATION é usada várias vezes durante uma associação, necessitando assim, de uma forma mais eficiente de atribuição de valores. Utiliza-se então, um número ou um "Object Identifier" que represente de forma única uma instância de operação, respectivamente, em um módulo ou de forma global à aplicação.

- **Exemplo de Criação de Tipos de Operações**

```
a) Operação-1  ::= OPERATION
                   ARGUMENT        Tipo-do-argumento-1
b) Operação-2  ::= OPERATION
                   RESULT          Tipo-do-resultado-2
c) Operação-3  ::= OPERATION
                   ERROR           Tipo-de-erro-3
                   LINKED          {operação-1, operação-2}
```

- **Exemplo de Instâncias da Macro OPERATION**

```
a) operação-1      OPERATION
                   ARGUMENT        Tipo-do-argumento-1
                   ::= 01
b) operação-2      OPERATION
                   RESULT          Tipo-de-resultado-2
                   ::= 02
c) operação-3   Operação-3 ::= {objectidentifier-3}
```

ERROR - Esta macro, por ser geralmente usada em resposta a alguma invocação que resultou em erro, contém apenas um tipo de parâmetro de erro. Sua notação de valor é idêntica à da macro OPERATION pelo fato dos erros poderem ser gerados várias vezes em uma mesma associação, precisando também de um número ou "Object Identifier" para sua notação de valor.

- **Exemplo de Criação de Tipos de Erros**

```
a) Erro-1      ::= ERROR
                   PARAMETER       Tipo-do-parâmetro-1
b) Erro-2      ::= ERROR
```

- **Exemplo de Instâncias da Macro ERROR**

```
a) erro-1          ERROR
                   PARAMETER       Tipo-do-parâmetro-1
                   ::= 01
b) erro-2    Erro-2  ::= {objectidentifier-2}
```

APPLICATION-CONTEXT - Define um protocolo de Aplicação, dentro de um contexto especificado. A Notação de tipo é formada por 4 tipos alternativos que podem ser combinados da forma já explicada anteriormente. O primeiro deles, *NonROelements* define os ASE's incluídos no contexto, excluíndo o ROSE; *Binding* define um tipo de operação BIND e um tipo de operação UNBIND; *ROelements* determina o ASE que dá suporte às operações remotas e se o relacionamento com a entidade par será simétrico ou assimétrico. Se simétrico, ambos poderão invocar operações. Se assimétrico, as seguintes opções são possíveis:

a) "INITIATOR CONSUMER OF" - Lista ASE's baseados na "RO-NOTATION" que contém operações que só podem ser invocadas pelo "initiator";
b) "RESPONDER CONSUMER OF" - Lista ASE's baseados na "RO-NOTATION" que contém operações que só podem ser invocadas pelo "responder".

O último dos tipos alternativos , *ABSTRACT-SYNTAX*, é um conjunto de "Object Identifiers" que é fornecido, como contexto de Apresentação, numa primitiva A-ASSOCIATE pela Entidade de Aplicação que o utilize. A notação de valor é definida pelo tipo "Object Identifier". A razão disto, está no fato do contexto de Aplicação ser único numa associação e precisar ser conhecido globalmente.

● Exemplo de Criação de Tipos de Contextos de Aplicação

a) Contexto-1

```
        ::= APPLICATION-CONTEXT
            APPLICATION SERVICE ELEMENTS {aCSE, rTSE}
            BIND     Operação-Bind-1
            UNBIND   Operação-Unbind-1
            REMOTE OPERATIONS        {rOSE}
            INITIATOR CONSUMER OF    {sASE}
            ABSTRACT-SYNTAXES     {objectidentifier1,
                                   objectidentifier2}
```

- **Exemplo de Instâncias da Macro Application-Context**

a) contexto-1 APPLICATION-CONTEXT
APPLICATION SERVICE ELEMENTS {aCSE, rTSE}
BIND Operação-Bind-1
UNBIND Operação-Unbind-1
REMOTE OPERATIONS {rOSE}
INITIATOR CONSUMER OF {sASE}
ABSTRACT-SYNTAXES {objectIdentifier1,
objectIdentifier2} ::= {objectidentifier-10}

APPLICATION-SERVICE-ELEMENT

É usada para definir um ASE. Se o ASE for um usuário do ROSE, é necessária a classificação de suas operações em :

a) "OPERATOR" - Lista de operações invocadas tanto por um "CONSUMER" quanto por um "SUPPLIER";
b) "CONSUMER INVOKES" - Lista de operações que podem ser invocadas pelo "CONSUMER";
c) "SUPPLIER INVOKES" - Lista operações invocadas pelo "SUPPLIER". Caso o ASE não seja usuário do ROSE, apenas um "Object Identifier" lhe será atribuido.

Após a definição do ROSE e sua notação, pode-se a partir destes, especificar os protocolos SASE's de forma mais simples e eficiente, através de um módulo (ASN.1) que defina suas interações e elementos. A seguir é mostrado um exemplo de especificação de um protocolo SASE imaginário chamado *SASE-exemplo*.

```
SaseExemplo { iso-exemplo (1) sintaxe-abstrata (0) }
DEFINITIONS ::=
BEGIN
IMPORTS  BIND, UNBIND, OPERATION, ERROR      FROM
         Remote-Operations-Notation
         {joint-iso-ccitt remote-operations (4) notation (0)}
```

```
        rOSE    FROM
        Remote-Operations-APDUs
        {joint-iso-ccitt remote-operations (4) apdus (1)}

        APPLICATION-CONTEXT, APPLICATION-SERVICE-ELEMENT, aCSE,
        aCSE-abstract-syntax  FROM
        Remote-Operations-Notation-extension
        {joint-iso-ccitt remote-operation (4) notation-extension (2)}

saseContext   APPLICATION-CONTEXT
              APPLICATION SERVICE ELEMENTS  {aCSE}
              BIND                          {Operação-bind}
              UNBIND                        {Operação-unbind}
              REMOTE OPERATIONS             {rOSE}
              OPERATIONS OF                 {sASE}
              ABSTRACT SYNTAXES             {aCSE-abstract-syntax,
                                             sASE-abstract-syntax}
               ::= { iso-exemplo (1) contexto-de-aplicação (1)}

sASE          APPLICATION-SERVICE-ELEMENT
              OPERATIONS                    {operação-01, operação-02}
              ::= { iso-exemplo (1) elemento-de-serviço (2)}

sASE-abstract-syntax   OBJECT IDENTIFIER
                       ::= {iso-exemplo (1) sintaxe-abstrata (0)}

Operação-bind  ::= BIND
                   ARGUMENT Tipo-de-argumento
                   RESULT   Tipo-de-argumento
Operação-unbind  ::= UNBIND
                     ERROR  Tipo-de-erro
operação-01    OPERATION
               ARGUMENT ANY
               RESULT   ANY
               ERRORS   {erro-01, erro-02}
operação-02    OPERATION
```

```
                ARGUMENT ANY

erro-01   ERROR
          PARAMETER  Tipo-do-parametro
          ::= 1

erro-02   ERROR
          ::= 2

.
.  Tipos ASN.1  que completam a especificação acima !
.
```

É importante notar que antes de usar uma macro já definida, é necessário importá-la de seu módulo de definição e que este módulo de definição exporte a mesma para que esteja disponível onde quer que seja pedido.

2.3. A Especificação dos Serviços do ROSE

Os serviços oferecidos pelo ROSE são:

Servico	tipo
RO-INVOKE	Não confirmado
RO-RESULT	Não confirmado
RO-ERROR	Não confirmado
RO-REJECT-U	Não confirmado
RO-REJECT-P	iniciado pelo provedor

tabela 2.1 - Serviços do protocolo ROSE

SERVIÇO RO-INVOKE

Este serviço é usado por um usuário do ROSE para fazer a invocação de uma operação a ser executada pelo usuário ROSE par. É formado apenas das primitivas *request* e *indication*. Seus parâmetros são:

Valor da Operação : Identifica a operação que esta sendo invocada. É fornecido por quem solicita o serviço.

Classe de Operação : Define se a operação será síncrona ou não, e em que ocasiões o iniciador deve receber relato do resultado. É fornecido por quem solicita o serviço.

Argumento : É uma informação adicional necessária a execução da operação. É também fornecido pelo iniciador.

Invoke-ID : Identifica o *request* do serviço e é usado para correlacionar este *request* com as respostas correspondentes (RO_RESULT, RO-ERROR, RO-REJECT-U, RO-REJECT-P) ou a invocação de operações relacionadas (RO-INVOKE). Este parâmetro tem que ser fornecido por quem solicita o serviço.

Linked-ID : Se estiver presente, a operação invocada é uma operação ligada e o parâmetro identifica a invocação da operação mãe correspondente. Seu valor é igual ao Invoke-ID da operação mãe.

Prioridade : Define a prioridade associada à transferência da APDU correspondente, com relação a outras PDU's que estão sendo trocadas.

SERVIÇO RO-RESULT

É usado para responder a uma primitiva RO-INVOKE.*indication* no caso da operação solicitada ter sido executada com sucesso. Só dispõe das primitivas *indication* e *request*. Seus parâmetros são:

Resultado : É o resultado de uma operação realizada com sucesso.

Invoke-ID : Trata-se do mesmo valor da operação que deu origem ao RO-RESULT.

Prioridade : Semelhante ao serviço anterior.

SERVIÇO RO-ERROR

Este serviço é usado para responder a uma primitiva RO-INVOKE.*indication* que não pode ser executada com sucesso. Dispõe, também, apenas das primitivas *request* e *indication*. Seus parâmetros são:

Valor do Erro : Identifica o erro que ocorreu durante a execução da operação.

Parâmetro do Erro : Fornece informação adicional sobre o erro.

Invoke-ID : É o mesmo do RO-INVOKE.*indication* que foi recebido pela entidade de protocolo par.

Prioridade : Semelhante ao correspondente no serviço anterior.

SERVIÇO RO-REJECT-U

O serviço RO-REJECT-U é usado para rejeitar uma requisição do usuário dos serviços ROSE se for detectado algum problema. Este serviço pode também ser usado por um usuário do ROSE para rejeitar uma resposta (RO-RESULT, RO-ERROR) vinda do usuário par. No entanto, para evitar violar as regras de seqüenciamento de outros ASE's em alguns contextos de Aplicação, o usuário pode preferir não usar o serviço RO-REJECT para rejeitar respostas. É formado apenas das primitivas *indication* e *request*. Seus parâmetros são:

Razão de Rejeição : Especifica o código da razão da rejeição de acordo com uma tabela predefinida.

Invoke-ID : é o mesmo da primitiva que está sendo rejeitada.

Prioridade : Semelhante ao parâmetro da primitiva usada no serviço anterior.

SERVIÇO RO-REJECT-P

é usado para avisar ao usuário ROSE de que um problema foi detectado pelo provedor. Este serviço é fornecido pelo provedor. Seus parâmetros são:

Invoke-ID : é o mesmo da primitiva rejeitada.

Parâmetros-Retornados : Este parâmetro contém os parâmetros do RO-INVOKE.*request*, RO-RESULT.*request*, RO-ERROR.*request* e RO-REJECT-U.*request*, se a APDU correspondente não pode ser transferida pelo provedor. Este parâmetro e o parâmetro Razão de Rejeição são mútuamente exclusivos.

Razão de Rejeição : é semelhante ao descrito no serviço anterior.

Com o conhecimento dos serviços do ROSE e dos demais protocolos das figuras 2.1 e 2.2, pode-se complementar as interações, mostradas anteriormente, por meio do mapeamento primitiva a primitiva ou notação a primitiva que ocorre no funcionamento normal de uma Entidade de Aplicação que seja definida nos contextos básicos apresentados para o ROSE. Veja as figuras 2.3 e 2.4 a seguir:

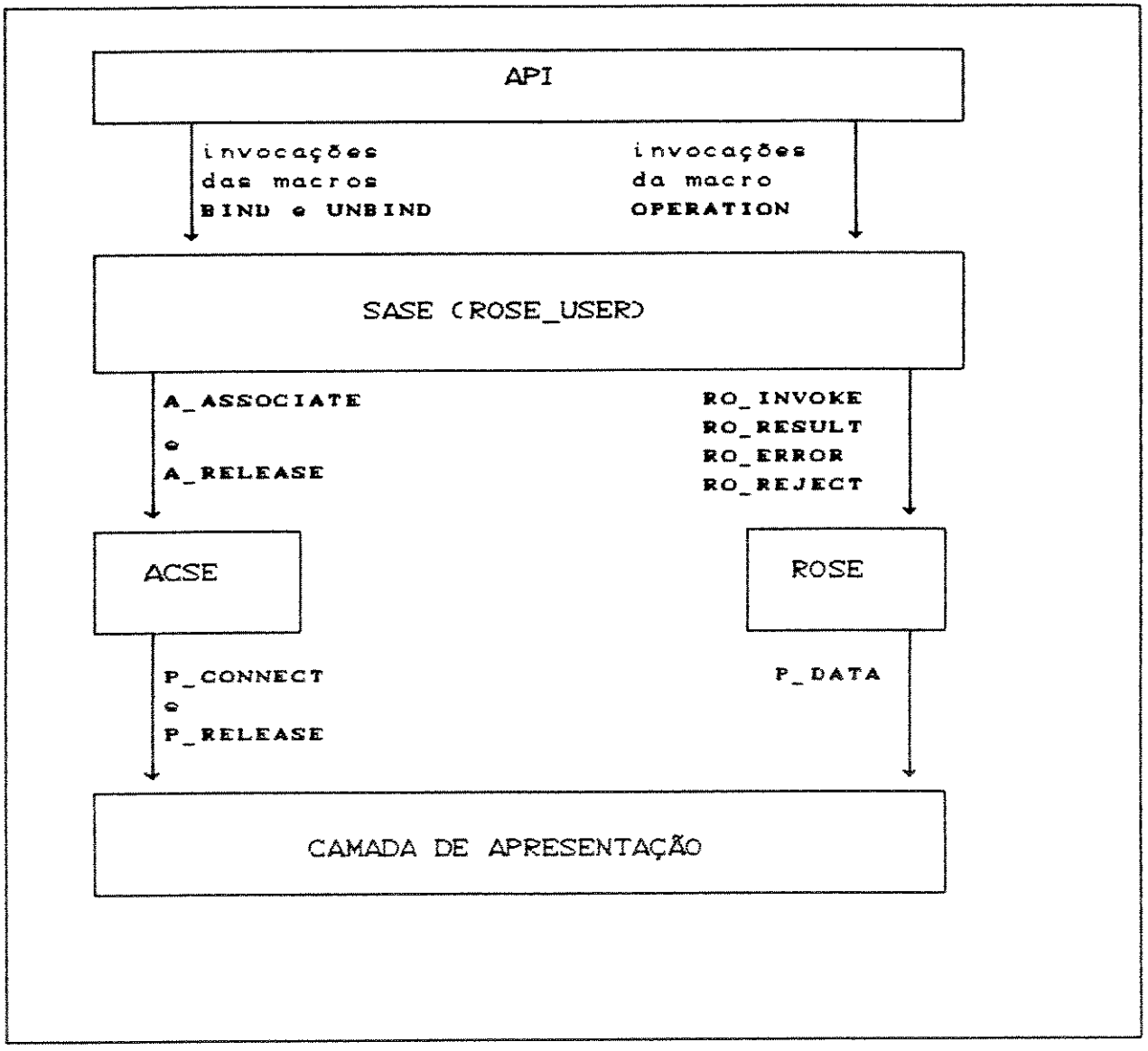


figura 2.3 - Esquema de Mapeamento em Contextos sem o RTSE

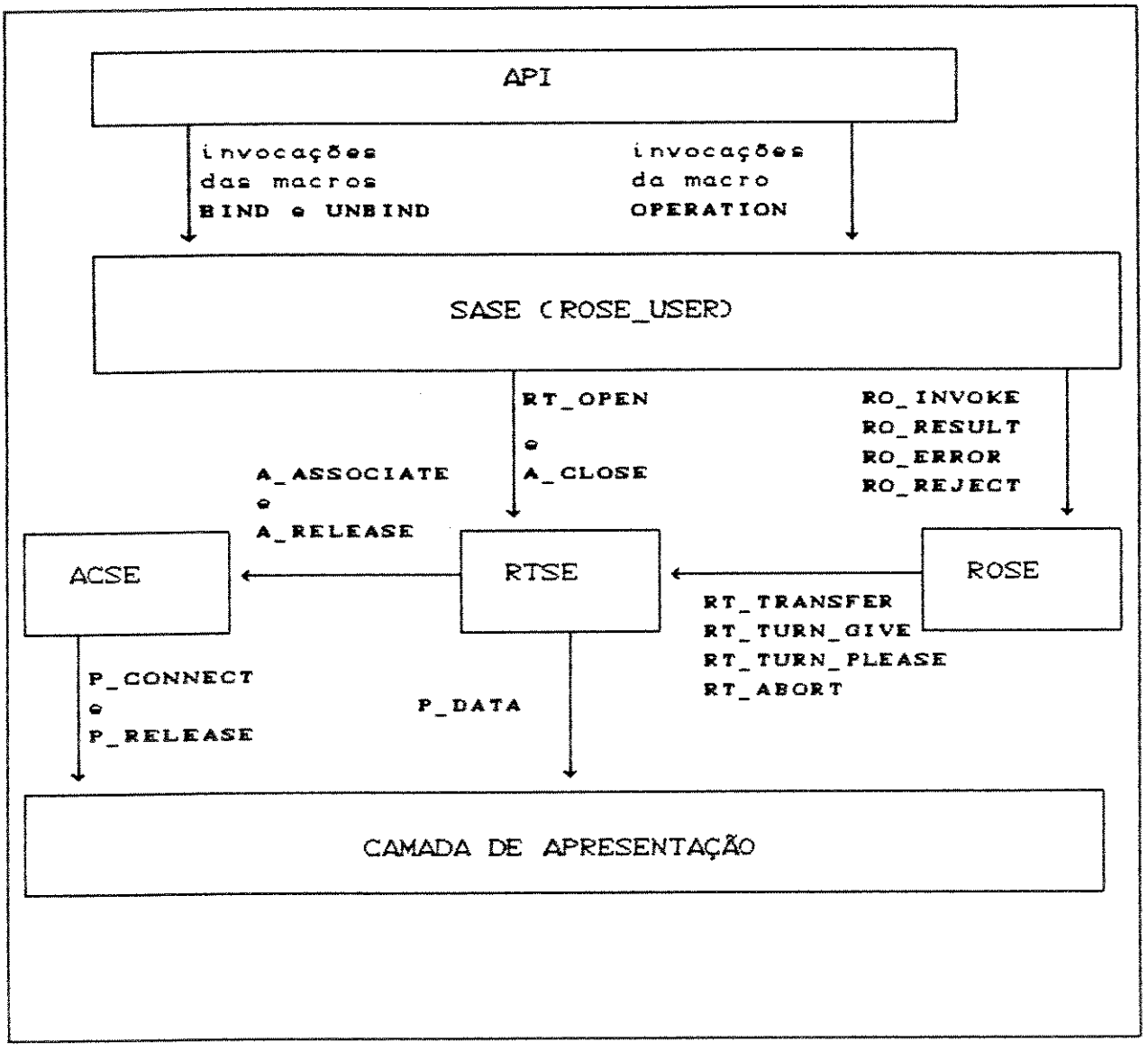


figura 2.4 - Esquema de Mapeamento em Contextos com o RTSE

2.4. A Especificação do Protocolo do ROSE

2.4.1. Elementos de Procedimento

O Protocolo do ROSE consiste dos seguintes elementos de procedimento:

a) Invocação : é usado por uma Entidade de Aplicação para requerer a execução de uma operação pela entidade par.

b) Retorno de Resultado : é usado para solicitar a transferência do resultado de uma operação efetuada com sucesso.

c) Retorno de Erro : é usado para transferir uma informação de erro no caso de uma operação não ter sido executada com sucesso.

d) Rejeição do Usuário : é usado por uma Entidade de Aplicação para rejeitar a invocação ou resposta de outra Entidade.

e) Rejeição do Provedor : é usado para informar o usuário do ROSE e a Máquina de Protocolo Par, que a Máquina local detectou algum problema.

Invocação

Utiliza a APDU ROIV que resulta de uma primitiva RO-INVOKE.*request*. Seus campos são:

Nome do Campo	Presença
Invoke-ID	M
Linked-ID	U
Valor da Operação	M
Argumento	U

tabela 2.2 - Campos da **APDU ROIV**

Este procedimento é dirigido pela chegada de uma RO-INVOKE.*request* ou pela chegada de uma APDU ROIV dentro de uma primitiva de transferência do tipo *indication*.

Retorno de Resultado

Utiliza a APDU RORS formada a partir de uma primitiva RO-RESULT. Seus campos são:

Nome do Campo	Presença
Invoke-ID	M
Resultado	U

tabela 2.3 - Campos da **APDU RORS**

Este procedimento é guiado pela chegada de uma primitiva RO-RESULT.*request* e pelo recebimento de uma APDU RORS dentro de uma primitiva de transferência do tipo *indication*.

Retorno de Erro

Utiliza a APDU ROER formada a partir de uma primitiva RO-ERROR. Seus campos são:

Nome do Campo	Presença
Invoke-ID	M
Valor do Erro	M
Parâmetro do Erro	U

tabela 2.4 - Campos da APDU ROER

Este procedimento é guiado pela chegada de uma primitiva RO-ERROR.*request* e pelo recebimento de uma APDU ROER dentro de uma primitiva de transferência do tipo *indication*.

Rejeição do Usuário

Usa a APDU RORJ formada a partir de uma primitiva RO-REJECT.*request*. Seus campos são:

Nome do Campo	Presença
Invoke-ID	M
Problem (choice of):	M
Invoke-problem	
Return-result-problem	
Return-error-problem	

tabela 2.5 - Campos da APDU RORJu

Este procedimento é guiado pela chegada de uma primitiva RO-REJECT-U.*request* e pelo recebimento de uma APDU RORJ dentro de uma primitiva de transferência do tipo *indication*.

Rejeição do Provedor

Este elemento de procedimento usa a mesma APDU RORJ do elemento anterior. Seus campos são:

Nome do Campo	Presença
Invoke-ID	M
Problem (choice of):	M
General-problem	

tabela 2.6 - Campos da APDU RORJ

Este procedimento é guiado pela chegada de uma APDU não aceitável, como dado de usuário, de uma primitiva de transferência do tipo *indication* e de uma APDU RORJ com o parâmetro de problema, como dado de usuário, numa primitiva de transferência do tipo *indication*.

Duas entidades de protocolo par comunicam-se entre si pela troca de APDU's. Tipicamente, uma PDU contém dados de usuário e PCI (Protocol Control Information) geradas pela camada em si. Desde que os parâmetros associados com uma primitiva de serviço só têm significado local, são normalmente definidos em termos de tipos de dados abstratos. As PDU's geradas por uma entidade de protocolo são passadas entre sistemas pares, precisando, portanto, evitar ambigüidades para que seus significados sejam comuns nos dois sistemas.

Para chegar a isto, as PDU's associadas com uma entidade de protocolo são definidas em documentos padrões de uma forma precisa, usando ou uma seqüência de bits específica ou tipos abstratos de dados (ASN.1).

As APDU's são mostradas a seguir:

```
Remote-Operations-APDUs
{joint-iso-ccitt remoteOperations(4)apdus(1)}

DEFINITIONS ::=
BEGIN
EXPORT rOSE, InvokeIDType;
```

```
IMPORTS OPERATION, ERROR FROM
                        Remote-Operation-Notation
                        {joint-iso-ccitt remoteOperation(4)notation(0)};

rOSE OBJECT IDENTIFIER  ::=
                        {joint-iso-ccitt remote Operations(4)aseID(3)}

-- APDU's
ROSEapdu      ::=  choice {[1] IMPLICIT ROIVapdu,
                        [2] IMPLICIT RORSapdu,
                        [3] IMPLICIT ROERapdu,
                        [4] IMPLICIT RORJapdu}

-- tipos de APDU's
ROIVapdu      ::=  SEQUENCE {
                        invokeID InvokeIDType,
                        linked-ID [0] IMPLICIT INTEGER OPTIONAL,
                        operation-value OPERATION,
                        argument ANY OPTIONAL
                        }

InvokeIDType::=  INTEGER

RORSapdu      ::=  SEQUENCE {
                        invokeID InvokeIDType,
                        result ANY OPTIONAL
                        }

ROERapdu      ::=  SEQUENCE {
                        invokeID InvokeIDType,
                        error-value ERROR,
                        parameter ANY OPTIONAL
                        }

RORJapdu      ::=  SEQUENCE {
                        invokeID CHOICE {InvokeIDType, NULL},
                        problem CHOICE {
                                [0] IMPLICIT GeneralProblem,
                                [1] IMPLICIT InvokeProblem,
                                [2] IMPLICIT ReturnResultProblem
                                [3] IMPLICIT ReturnErrorProblem}}

--  Razão de Rejeição das APDU's

GeneralProblem     ::= INTEGER {
                                unrecognisedAPDU(0),
                                mistypedAPDU(1),
                                badlyStructuredAPDU(2)}


InvokeProblem      ::= INTEGER {
                                duplicateInvocation(0),
                                unrecognisedOperation(1),
                                mistypedArgument(2)
```

```
                                    resourceLimitation(3)
                                    initiatorReleasing(4)
                                    unrecognizedLinkedID(5)
                                    linkedResponseUnexpected(6)
                                    unexpectedChildOperation(7)}

ReturnResultProblem ::= INTEGER {
                                    unrecognisedInvocation(0),
                                    resultResponseunexpected(1),
                                    mistypeResult(2)}

ReturnErrorProblem  ::= INTEGER {
                                    unrecognisedInvokation(0),
                                    erroResponseUnexpected(1),
                                    unrecognisedError(2),
                                    unexpectedError(3),
                                    mistypedParameter(4)}
```

2.4.2. Máquina de Estados do ROSE

O Documento ISO 9072-2 [2] apresenta uma tabela de eventos/estados que mostra o relacionamento entre o estado de uma associação, os eventos de entrada que ocorrem para o protocolo, as ações tomadas e os estados finais desta associação. Esta tabela não é, obviamente, uma definição completa de implementação da máquina de protocolos do ROSE, mas fornece uma especificação mais precisa dos elementos de procedimento do protocolo.

A Implementação feita, no contexto do SISDI_MAP, baseou-se nesta especificação proposta da tabela de eventos/estados da máquina de estados finitos do ROSE. A seguir, são mostradas várias tabelas: tabs 2.7,2.8, 2.9, 2.10, 2.11, 2.12, 2.13 e 2.14 que relacionam os elementos componentes da ROPM e ROPM-TR, que são r espectivamente a máquina de protocolos que descreve o relacionamento com um usuário dos serviços do ROSE e uma máquina que descreve o relacionamento com o provedor de serviços que o ROSE utiliza.

Nome Abreviado	Descrição
STA01	Não associado
STA02	Associado

tabela 2.7 - Relação de Estados da ROPM

Nome Abreviado	Descrição
STA10	Não associado
STA20	Associado, com Token e Sem Transferir
STA21	Associado, com Token e Transferindo
STA22	Associado, sem Token e Sem Transferir
STA23	Associado, sem Token e Transf. solic.
STA100	Não associado - sem RTSE no contexto
STA200	Associado - sem RTSE no contexto

tabela 2.8 - Relação de Estados da ROPM-TR

Nome Abreviado	Origem	Descrição
AA_ESTAB	RTSE ACSE	primitiva RT_OPEN resp(+)/conf(+) primitiva A_ASSOCIATE resp (+) ou primitiva A_ASSOCIATE conf (+)
RO_INV_REQ	ROSE-user	primitiva RO_INVOKE request
RO_RES_REQ	ROSE-user	primitiva RO_RESULT request
RO_ERR_REQ	ROSE-user	primitiva RO_ERROR request
RO_RJU_REQ	ROSE-user	primitiva RO_REJECT_U request
ROIV	ROPM-par	RO_INVOKE APDU na TRANS_IND
RORS	ROPM-par	RO_RESULT APDU na TRANS_IND
ROER	ROPM-par	RO_ERROR APDU na TRANS_IND
RORJu	ROPM-par	RO_REJECT user APDU na TRANS_IND
RORJP	ROPM-par	RO_REJECT prov. APDU na TRANS_IND
APDUUA	ROPM-par	APDU não aceitável em TRANS_IND
TRANS_IND	ROPM	indicação de chegada de transf.
TRANS_REQ	ROPM	pedido de transferência de APDU
P_DATA_IND	PS-PROV.	primitiva P_DATA indication
RT_TR_IND	RTSE	primitiva RT_TRANSFER indication
RT_TP_IND	RTSE	primitiva RT_TURN_PLEASE indicat.
RT_TR_CNF_POS	RTSE	primitiva RT_TR_CNF(+) indication
RT_TR_CNF_NEG	RTSE	primitiva RT_TR_CNF(-) indication
RT_TG_IND	RTSE	primitiva RT_TURN_GIVE indication
AA_AB_REQ	ROPM	pedido de aborto de associação
AA_AB_IND	ROPM-TR	indicação de aborto de associação
ABORT_IND	RTSE ACSE	primitiva RT_U_ABORT indication primitiva A_ABORT ou A_P_ABORTind
AA_REL	RTSE ACSE	primit. RT_CLOSE resp(+)/conf(+) primit. A_RELEASE resp(+)/conf(+)

tabela 2.9 - Relação de Eventos de Entrada para ROPM e ROPM-TR

Nome Abreviado	Destino	Descrição
RO_INV_IND	ROSE-user	primitiva RO_INVOKE indication
RO_RES_IND	ROSE-user	primitiva RO_RESULT indication
RO_ERR_IND	ROSE-user	primitiva RO_ERROR indication
RO_RJU_IND	ROSE-user	primitiva RO_REJECT_U indication
RO_RJP_IND	ROSE-user	primitiva RO_REJECT_P indication
ROIV	ROPM-par	RO_INVOKE APDU na TRANS_REQ
RORS	ROPM-par	RO_RESULT APDU na TRANS_REQ
ROER	ROPM-par	RO_ERROR APDU na TRANS_REQ
RORJu	ROPM-par	RO_REJECT user APDU na TRANS_REQ
RORJP	ROPM-par	RO_REJECT prov. APDU na TRANS_REQ
TRANS_REQ	ROPM-TR	pedido interno de transf. de APDU
TRANS_IND	ROPM	indicação de pedido de transf.
P_DATA_REQ	PS-PROV.	primitiva P_DATA request
RT_TR_REQ	RTSE	primitiva RT_TRANSFER request
RT_TP_REQ	RTSE	primitiva RT_TURN_PLEASE request
RT_TG_REQ	RTSE	primitiva RT_TURN_GIVE request
AA_AB_REQ	ROPM-TR	pedido de aborto de associação
AA_AB_IND	ROPM	indicação de aborto de associação
ABORT_REQ	RTSE ACSE	primitiva RT_U_ABORT request primitiva A_ABORT request

tabela 2.10 - Relação de Ações e seus Destinatários

Codigo	Nome e Descrição
p1	APDU não aceitável não é RORJ APDU e o nº de rejeições não excedeu o valor máximo especificado.
p2	O Token está inicialmente com a ROPM_TR local.

tabela 2.11 - Relação de Predicados (condições) Usados

	STA01	STA02
AA_ESTAB	STA02	
RO_INV_REQ		ROIV STA02
RO_RES_REQ		RORS STA02
RO_ERR_REQ		ROER STA02
RO_RJU_REQ		RORJU STA02
ROIV		RO_INV_IND STA02
RORS		RO_RES_IND STA02
ROER		RO_ERR_IND STA02
RORJU		RO_RJU_IND STA02
RORJP		RO_RJP_IND STA02
APDUUA		p1: RORJP STA02 ¬p1: AA_AB_REQ STA01
AA_AB_IND		STA01
AA_REL		STA01

tabela 2.12 - Tabela Eventos/Estados da ROPM

	STA10	STA20	STA21	STA22	STA23
AA_ESTAB	p2: STA20 ¬P2: STA22				
TRANS_REQ		RT_TR_REQ STA21		RT_TP_REQ STA23	
RT_TR_CNF POS			STA20		
RT_TR_CNF NEG			RO_RJP_IND STA20		
RT_TR_IND				TRANS_IND STA22	TRANS_IND STA23
RT_TP_IND		RT_TG_REQ STA22	STA21		
RT_TG_IND				STA20	RT_TR_REQ STA21
AA_AB_REQ		ABORT_REQ STA10	RO_RJP_IND ABORT_REQ STA10	ABORT_REQ STA10	RO_RJP_IND ABORT_REQ STA10
ABORT_IND		AA_AB_IND STA10	RO_RJP_IND AA_AB_IND STA10	AA_AB_IND STA10	RO_RJP_IND AA_AB_IND STA10
AA_REL		STA10	RO_RJP_IND STA10	STA10	RO_RJP_IND STA10

tabela 2.13 - Tabela de Eventos/Estados da ROPM_TR

	STA100	STA200
AA_ESTAB	STA200	
TRANS_REQ		P_DATA_REQ STA200
P_DATA_IND		TRANS_IND STA200
AA_AB_REQ		ABORT_REQ STA100
ABORT_IND		AA_AB_IND STA100
AA_REL		STA100

tabela 2.14 - Tabela de Eventos/Estados da ROPM_TR (sem RTSE)

3.0. Especificação da Implementação

A implementação de um protocolo, em princípio, é uma tarefa que pode ser feita tanto a nível de "hardware" quanto a nível de "software". A tendência atual é implementar apenas as camadas mais inferiores do modelo OSI, ou seja, as camadas: Física, de Enlace e de Rede, em "hardware", usando para isso, sistemas com microprocessadores e até pastílhas de circuito integrado. As camadas de alto nível: Transporte, Sessão, Apresentação e Aplicação; no entanto, são, via de regra, implementadas em "software". O ROSE, por ser um subprotocolo da camada de Aplicação, é também implementado em "software".

Antes de implementar um protocolo, é fundamental discutir e definir alguns aspectos da especificação que não foram definidos explicitamente na documentação e são conhecidos como "local implementation matters". Tais aspectos foram deixados em aberto para que as decisões tomadas pelo implementador reflitam peculiaridades do sistema onde rodará a implementação e/ou para que as decisões possibilitem níveis desejados de desempenho para a operação do protocolo. No protocolo das operações remotas, são exemplos de "local implementation matters": a implementação das interfaces, a definição de valores de constantes, o tratamento de prioridades dos serviços do ROSE e a forma como se indicará ao ROSE que uma determinada associação foi estabelecida ou liberada.

Outra definição importante que deve ser feita antes da implementação propriamente dita, é a da integração ao sistema local em que vai operar o código final. Segundo o exposto em [16], isto é feito com base nos seguintes ítens:

- minimizar custo de serviços de comunicação;
- maximizar vazão nas conexões utilizadas e
- minimizar a utilização de recursos do sistema dedicados à comunicação.

No nosso caso, os dois primeiros ítens são irrelevantes em virtude do fato do SISDI_MAP apenas simular um ambiente de rede e não ter preocupações com custos, nem com taxas de vazão, ficando apenas a última para nortear a integração.

As integrações mais comuns são:

a) implementação como processo de usuário;
b) implementação no núcleo do Sistema Operacional ou
c) implementação num processador "Front-End".

Conforme foi visto no capítulo 1, o SISDI_MAP também já tem sua estrutura de integração definida como um conjunto de vários processos de usuário, coordenados por um núcleo dedicado que está sobre o sistema operacional da máquina, o que leva a escolha natural da opção a.

Após a definição dos detalhes acima, usa-se um modelo de funcionamento interno da camada/subcamada para implementar o protocolo em uma linguagem de programação definida, no caso C. Neste ponto, o implementador não fica tão dependente do sistema, no qual está contido o protocolo, pelo fato do funcionamento interno das camadas/subcamadas, projetadas de acordo com o RM-OSI/ISO, serem independentes umas das outras. A dependência que existe, está limitada apenas ao interfa-ceamento e ao cálculo de algumas constantes de temporização.

Existe um modelo geral de implementação muito conhecido e que é apresentado em [16] como sugestão por ser válido para qualquer implementação realizada em "software". O esquema em blocos deste modelo é apresentado a seguir. Neste, cada módulo corresponde univocamente a uma transição, procedimento, primitiva ou evento das interfaces definidas na especificação.

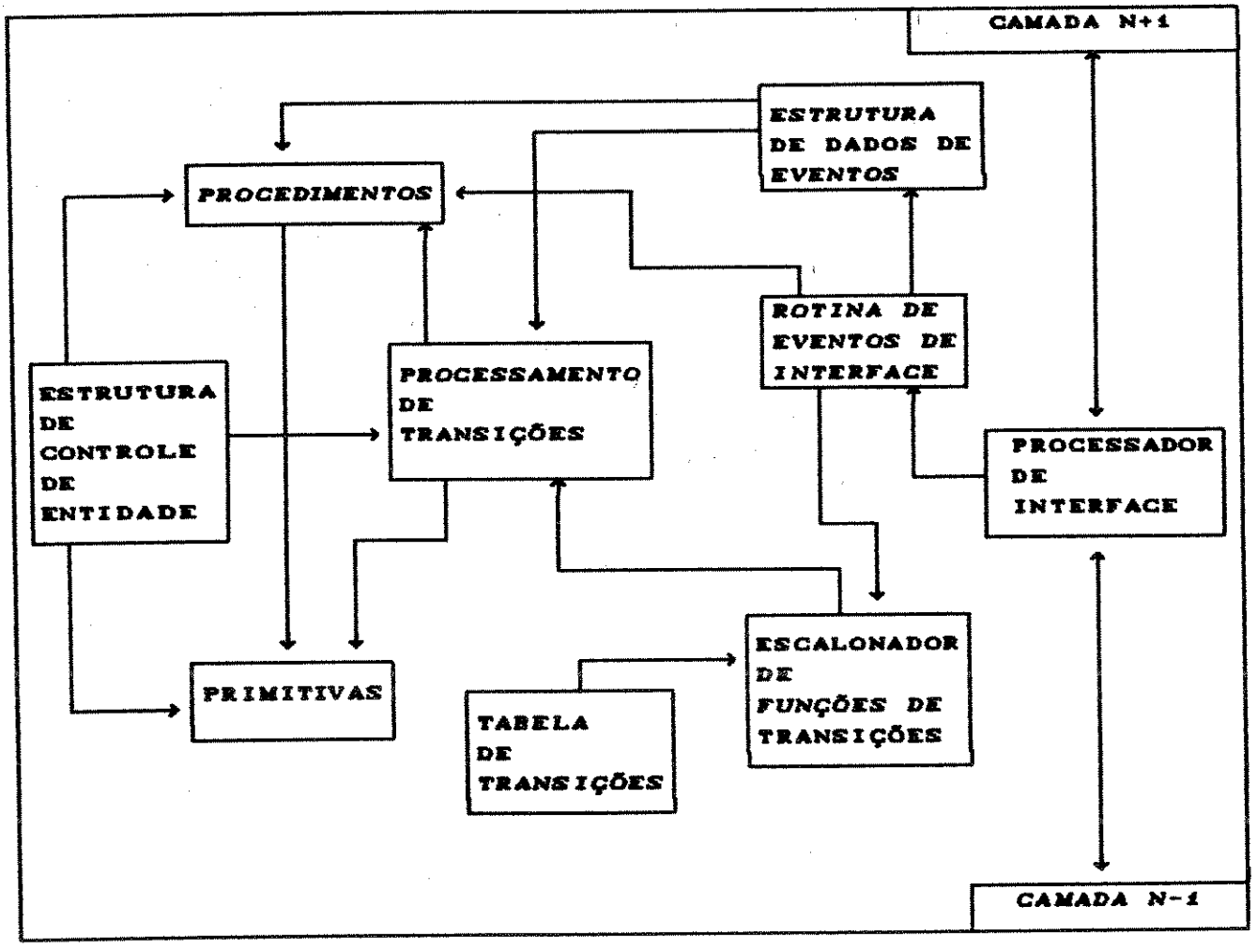


figura 3.1 - Modelo de implementação para a camada N

O funcionamento é simples; ao chegar um evento, ele é identificado pelo Processador de Interface para determinar qual a entidade (associação) de destino que deverá processá-lo. Em seguida, o processador de interface escalona o evento para atendimento pelas rotinas de Eventos da Interface. A rotina de eventos de interface apropriada então constrói uma Estrutura de Dados com informações sobre o evento de interface particular e chama o Escalonador de funções de Transição.

De posse do tipo de evento de interface e da identificação da entidade-destino, o escalonador examina o estado dessa entidade na estrutura de controle de entidades. O estado presente da entidade e o tipo de evento servem de índice na tabela de transições para obter uma lista de transições possíveis, em ordem de prioridade. O escalonador então chama o módulo responsável para processar o tipo de transição com a prioridade mais alta. Se nenhuma transição é aplicável, uma indicação de erro é enviada de volta e o evento descartado. Ç

O módulo chamado pode, por sua vez, em função das ações especificadas para a transição a ser processada, chamar procedimentos, invocar primitivas ou gerar eventos para as camadas adjacentes.

A geração desses eventos é feita pelas rotinas de eventos de interfaces que constróem as estruturas de dados associadas e chamam o processador de interface para despachar os eventos para a camada apropriada.

O modelo adotado é baseado no modelo geral sugerido, mas é um pouco diferente na estrutura. A razão desta diferença é explicitar ao máximo, no esquema, o que os documentos de padronização do ROSE mostram. Vale salientar, no entanto, que todas as funções e interações do modelo anterior são mantidas, só que agora subdivididas em grupos diferentes. O novo esquema é mostrado a seguir:

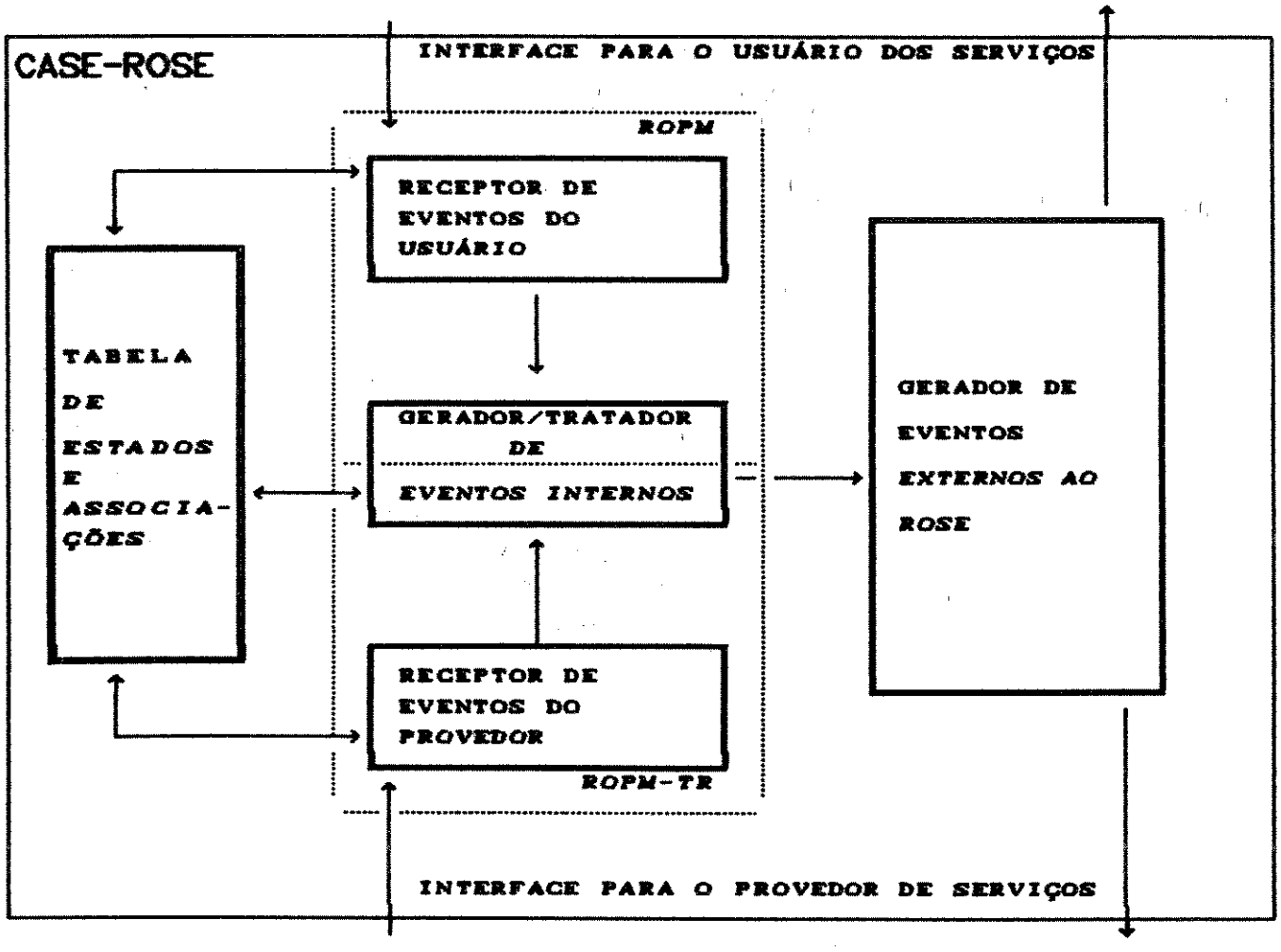


figura 3.2 - Modelo para implementação do ROSE

Neste esquema, o Processador de Interface está dividido funcionalmente em receptor de eventos do provedor, receptor de eventos do usuário e gerador de eventos externos ao ROSE. A Estrutura de Controle de Entidades está contida na tabela de estados e associações. Os Procedimentos, Primitivas e a Rotina de Eventos de Interface estão implícitos no gerador de eventos externos ao ROSE. O Escalonador de Funções de Transições e o Processamento de Transições estão implícitos na lógica do relacionamento dos blocos acima. O gerador/tratador de eventos internos se enquadra no bloco correspondente a Procedimentos em virtude de ser uma ocorrência característica interna, representada por um conjunto de ações.

A partir das definições relativas à integração e do esquema criado para o ROSE, dois caminhos podem ser seguidos: a codificação direta em C ou a utilização de alguma ferramenta de geração automática. No princípio do projeto SISDI_MAP, foi adotado o primeiro caminho por não se dispor de nenhum "software" adequado a codificação automática. Quando se decidiu expandir o projeto para que o mesmo suportasse as camadas de Sessão, Apresentação e o ASE-ROSE, já estava disponível um pacote chamado EPOS, usado para projetos de "hardware" e "software" e com o adicional de gerar toda a documentação do projeto, durante o desenvolvimento do mesmo. Decidiu-se, então fazer as extensões ao SISDI_MAP através do EPOS, visando-se obter experiência na geração automática de código.

3.1. Especificação de Requisitos Utilizando o EPOS-R

Antes de iniciar qualquer projeto com o uso do EPOS, é preciso formalizar todos os requisitos e restrições inerentes ao mesmo para que um outro módulo do pacote, o EPOS-A, possa fazer um teste de conformidade automático entre o que se deseja e o que foi realmente implementado.

Uma especificação em EPOS-R consiste de duas partes: a descrição do problema a ser abordado e a solução conceitual que é a definição da estratégia para solução do mesmo.

A descrição do problema trata dos seguintes aspectos:

a) descrição da estrutura do problema;
b) descrição dos requisitos funcionais;
c) descrição das restrições;
d) descrição dos requisitos lógicos e
e) especificação e definição dos têrmos importantes.

No caso, o problema é a implementação do protocolo e seus "local implementation matters" associados.

A solução conceitual preocupa-se com aspectos do tipo:

a) as possíveis soluções para o problema;
b) os requisitos dependentes da solução;
c) componentes da solução escolhida;
d) componentes lógicos dependentes da solução e
e) definição dos têrmos importantes utilizados.

A solução adotada foi a implementação direta da lógica proposta nas tabelas de eventos/estados apresentadas no documento ISO 9072-2, juntamente com a definição mais conveniente ao SISDI_MAP dos "local implementation matters".

A construção de uma especificação no EPOS-R é formada pelos elementos definidos abaixo:

a) ítens classificados em estrutura decimal, definidos pelo usuário;
b) sequência de textos verbais;
c) elementos formais;
d) glossário.

A especificação pode também ser interpretada como um conjunto de entradas básicas formadas por um cabeçalho e um corpo, onde o cabeçalho é formado por um elemento do tipo a associado a um texto entre aspas que representa o nome da entrada e o corpo é uma combinação dos demais elementos relacionados.

O que é chamado de texto verbal é simplesmente uma sequência de caracteres, de formatação ou não, colocados entre aspas. Os elementos formais por sua vez são formações do tipo:

```
* REQUIREMENT i(j): <category>
      "descrição do requisito em linguagem natural";
* CONSTRAINT i(j): <category>
      "descrição da restrição.";
```

```
* TERM <nome do termo>  ou
* DECISION-PROCESS <nome>
  CONDITIONS:
              condição 1(valor1,...,valorN1)
              condição 2(valor1,...,valorN2)
              ...
              condição N(valor1,...,valorNm).
  OPERATIONS:
              operação 1,
              operação 2,
              ...
              operação N.
  RULES:
     (condição 1,...,condição N;<ordem de execução das op.>)

  COMMENT:

     "comentário pertinente ao processo"
```

As palavras chaves REQUIREMENT e CONSTRAINT representam, respectivamente, requisito e restrição. Após cada uma delas vem um ou dois números que as representam unicamente na especificação. Pode-se também agrupar os elementos formais em "categories" que são colocadas imediatamente antes do texto descritivo de cada uma.

São estes elementos, os responsáveis pela conexão lógica existente entre o EPOS-R e o EPOS-S e, entre a definição do problema e a solução conceitual.

O terceiro elemento formal representa uma referência a um têrmo previamente definido no glossário. Isto implica na inclusão da definição do têrmo no local da referência quando da geração da documentação correspondente. Este elemento é responsável pela conexão lógica entre o glossário e demais componentes do EPOS-R.

Os DECISION-PROCESS's são elementos formais que permitem a criação de tabelas organizadas na documentação gerada automaticamente. É importante salientar que os mesmos precisam estar associados a um REQUIREMENT para entrarem na análise de conformidade. O nome representa um processo geral que está sujeito a um conjunto de condições relacionadas em CONDITIONS e que pode disparar uma ou mais operações de um conjunto relacionado em OPERATIONS. As regras que definem quais operações são disparadas e em que condições, são mostradas em RULES. Em RULES, para cada linha existe um conjunto de valores associados às condições e o seguinte conjunto de símbolos:

X <=> a operação correspondente a posição deste símbolo deve ser disparada;

1 <=> a operação correspondente a posição deste símbolo deve ser a primeira a ser disparada;

2 <=> a operação correspondente a posição deste símbolo deve ser a segunda a ser disparada;

- <=> a operação correspondente a posição deste símbolo não deve ser disparada.

A conexão lógica entre dois módulos ou submódulos precisa necessariamente da definição de um elemento formal em um delese da utilização de um elemento formal do tipo FULFILS no outro. Sua sintaxe é:

FULFILS: REQUIREMENT i(j) ou
FULFILS: CONTRAINT i PARTLY

A palavra chave PARTLY serve para indicar que o requisito ou restrição especificado é realizado por mais de um elemento.

A especificação de requisitos gerada pelo EPOS para a implementação do ASE ROSE é apresentada em [17] e [18].

3.2. Especificação Formal da Implementação em EPOS-S

Embora a especificação em EPOS-R tenha alguns elementos formais, a formalização da implementação a rigor só se dá na especificação em EPOS-S pelo fato desta dispor de uma sintaxe e semântica rígidamente estruturadas.

A especificação em EPOS-S é formada por um grupo de objetos de projeto, hierarquicamente estruturados em níveis e com interações bem definidas entre si. A hierarquia existente entre os níveis é causada pelo diferente grau de abstração existente em cada nível, ou seja, inicia-se a especificação com objetos mais abstratos que se subdividem em outros mais refinados que, por sua vez, são também subdivididos até se chegar a objetos que representem puramente segmentos de código de uma linguagem de programação qualquer. É fácil perceber que pode-se perfeitamente desenvolver todo um sistema sem se preocupar com a linguagem, simplesmente não se definindo os objetos do último nível hierárquico. Desta forma, pode-se classificar a linguagem EPOS-S como independente de linguagem final.

As interações existentes entre os objetos podem ser por fluxo de dados ou por fluxo de controle. O fluxo de dados corresponde à passagem de parâmetros ou exportação/importação de dados. O fluxo de controle indica a ordem de execução das ações e/ou operações.

Os componentes sintáticos usados são:

a) palavras chaves;
b) identificadores;
c) constantes;
d) textos;
e) caracteres especiais e símbolos.

3.2.1. Os Objetos de Projeto

A definição de um objeto começa com uma palavra chave que indica o tipo do objeto e o nome arbitrário que identifica de forma única o objeto na especificação. O fim da definição é indicado com uma palavra formada pelo tipo do objeto mais o sufixo END. No corpo de um objeto existem várias definições possíveis, conforme o esquema abaixo:

```
<TIPO>                <nome do objeto>
<DESCRIÇÃO>
<DECOMPOSIÇÃO>
<ESPECIFICAÇÃO>
<RELACIONAMENTO>
<TIPO>END
```

Figura 3.3 Modelo Geral de uma Especificação em EPOS S

Os tipos de objetos existentes são: Action, Data, Condition, Interface, Event ou Device.

a) **ACTION** - Descreve atividades nas quais os dados são processados. Pode representar um processamento puramente funcional, não relacionado com uma realização específica ou pode representar o processamento de dados por um programa, periférico ou pessoa. Uma action pode ser de quatro subtipos:

- **Module** - Descreve subsistemas e não é envolvido pelo fluxo de controle;
- **Task** - Descreve uma tarefa que pode ser disparada por um evento;
- **Procedure** - Descreve uma subrotina com função específica;
- **Macro** - Representa uma especificação que é simplesmente substituida pelo nome da action macro no corpo da

especificação.

b) **DATA** - Descreve a informação processada durante a fase de projeto, tal como variáveis, tipos de dados ou constantes.

c) **CONDITION** - Representa condições que influenciam o fluxo de controle. É uma condição lógica : "true" ou "false".

d) **INTERFACE** - Descreve a troca de dados entre o sistema e sua estrutura. Têm sua definição muito parecida com a dos objetos DATA;

e) **EVENT** - São usados para descrever eventos que são representados por sinais binários, e que influenciam a ordem das ações.

f) **DEVICE** - Define dispositivos ou periféricos responsáveis pela execução física de algumas ações definidas.

3.2.2. Filosofias de Especificação para Implementação

Dependendo das características do projeto, alguns métodos de especificação são melhores do que outros. O EPOS permite sete métodos de especificação, seis deles com suporte "on line" e um independente de orientação, o que deixa o projetista inteiramente livre para criar um método misto. São eles:

a) Projeto Orientado por Funções: Divide-se, inicialmente, o sistema em funções, determina-se uma ordem de execução e, depois representa-se cada função por ações do tipo PROCEDURE e os demais objetos apenas complementam aspectos como parâmetros trocados entre as funções, condições no fluxo de controle, etc.

b) Projeto Orientado por Eventos: É aplicado a sistemas que exigem resposta rápida a ocorrência de eventos. Define um sistema como um conjunto de ações do tipo TASK e funções a serem executadas. As funções são ações do tipo PROCEDURE e os dados são acrescen-

tados posteriormente.

c) Projeto Orietado por Módulos: O sistema é subdivididos em subsistemas e cada subsistema é subdividido novamente em outros subsistemas. Este processo continua sucessivamente até um subsistema indivisível que é especificado em detalhes. Este método é baseado principalmente na ação tipo MODULE.

d) Projeto Orientado por Fluxo de Dados: Inicia-se com a informação que deve ser armazenada, combinada e manipulada. A descrição nas transformações necessárias implica nas de ações responsáveis por sua execução. Essas ações são do tipo PROCEDURE.

e) Projeto Orientado por Estrutura de Dados: As estruturas de entrada e saída do problema são definidas e analisadas primeiramente. Depois, um programa é derivado para fazer as transformações convenientes na entrada para se obter a saída esperada. Neste método o objeto tipo DATA é privilegiado.

f) Projeto Orientado Por Dispositivos: É usado para projetos de "hardware". Os componentes são definidos através do objeto DEVICE, e subdivididos sucessivamente. As Subdivisões terminais são representadas pelos demais objetos.

g) Projeto Misto: É qualquer outro método que não se enquadre nos descritos anteriormente. Não existe orientação "on line" durante o projeto, nem objetos necessariamente privilegiados em relação a outros.

Das características apresentadas para cada método de projeto e da forma como já estava estruturado o SISDI_MAP, escolheu-se a filosofia do projeto MISTO. Esta escolha se deu, mais precisamente, pelas seguintes razões:

. a linguagem adotada é C;
. limitações no mapeamento de alguns objetos em código final;
. características próprias da implementação de um protocolo;

. requisitos específicos do SISDI_MAP;

O fato do primeiro não ser escolhido, se deu porque houve no projeto uma divisão funcional do problema em módulos com funções independentes. Havendo, portanto, a utilização da ação MODULE como objeto de partida para todo o projeto.

A princípio, o método orientado a eventos parece o mais adequado ao projeto, em virtude das características intrínsecas de um protocolo. O impedimento que o eliminou foi basicamente causado por limitações na geração de código automática, ou seja, o elemento de projeto principal, EVENT, não era mapeado automaticamente para uma estrutura ou procedure da linguagem C. Além disto, o Sistema SISDI_MAP é um projeto que não tem a rigor as exigências de um "software" para tempo real.

A metodologia orientada a módulos implica na filosofia de transparência de informação, ou seja, o que um módulo conhece precisa ser importado para estar disponível a um outro fora de seu espaço de visibilidade. No projeto adotado não havia necessidade desta limitação.

Os demais não se aplicavam pela própria característica de um protocolo, ou seja, existem vários eventos de entrada e vários eventos de saída; a estrutura de dados para a comunicação era muito complicada para ser especificada no EPOS.

3.2.3. A Implementação

A implementação através dos objetos discutidos na seção anterior, tomou por base o modelo definido para o funcionamento interno apresentado na figura 3.2. Primeiro criou-se um módulo principal, chamado CASE_ROSE, a partir do qual todo o projeto é definido. Sabendo-se da necessidade de fazer-se inclusões de bibliotecas de definições do SISDI_MAP e do compilador usado e, também, para maior clareza da especificação, o módulo principal foi dividido em dois outros, ROSE_PROTOCOL e ROSE_INCLUDES, que representam, respectivamente, o programa que constitui o processo ROSE e o con-

junto de arquivos de bibliotecas usados.

O módulo ROSE_INCLUDES concentra submódulos que representam individualmente os "includes": STDIO, SISDI1, SISDI2, ROSE_DATA e RTSE_DATA. A definição precisa destes arquivos é feita no apêndice A. Pelo fato do programa e seus "includes" estarem em espaços de visibilidade diferentes, os módulos dos mesmos devem ser exportados pelo ROSE_INCLUDES e importados pelo ROSE_PROTOCOL para que, na geração automática, apareçam no código final na forma de comandos de inclusão do pré-processador do compilador. A Figura 3.4 foi gerada automaticamente através do EPOS-D e mostra o esquema hierárquico dos módulos discutidos.

O ROSE_PROTOCOL é o módulo que abrange a implementação propriamente dita, pois tem como subdivisão uma ACTION TASK chamada PROC_ROSE a partir da qual se descreve toda a lógica funcional da máquina de protocolos do ROSE.

A decomposição da ação PROC_ROSE é mostrada a seguir na forma de um diagrama "dataflow" também gerado no EPOS-D. São mostrados dois diagramas, cada um correspondendo a um contexto diferente, ou seja, com ou sem RTSE (Figuras 3.5 e 3.6).

Estes esquemas representam os programas principais das especificações de implementação feitas. Neles, temos as ações representadas por retângulos e as condições por losângos. A primeira ação é responsável pela definição das variáveis e tipos locais e está definida em [19] e [20]. Logo depois, um "loop" infinito faz com que o programa fique sempre verificando as condições MSG1_FROM_SASE, MSG2_FROM_SASE, MSG_FROM_RTSE (ou MSG_FROM_PRES), CONTROL_PORT_1 e CONTROL_PORT_2 que, se verdadeiras, indicam a presença de mensagens na interface com o usuário ou na interface com o provedor em uma porta de controle.

As duas primeiras, MSG1_FROM_SASE e MSG2_FROM_SASE, representam a interface com o usuário do serviço, uma de maior prioridade que a

outra, a terceira com o provedor do servico, a quarta é usada na indicação de estabelecimento de associação e a quinta na indicação de liberação de associação. Todas as condições estão definidas em [19] e [20].

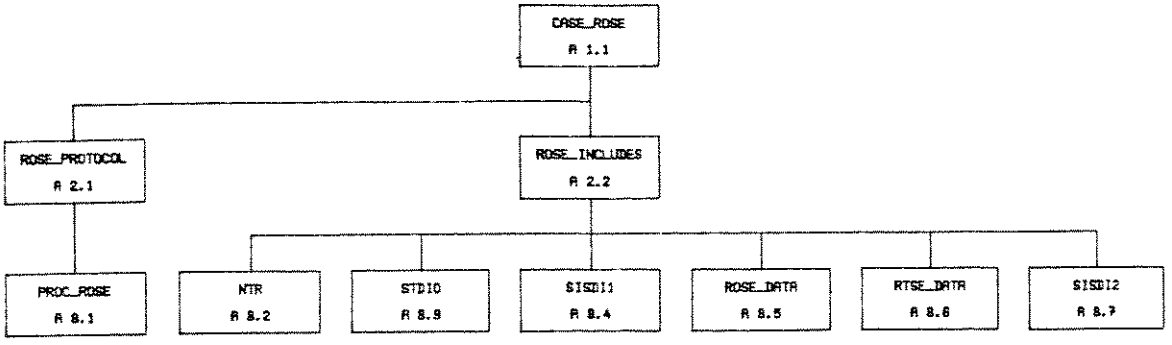


figura 3.4 - Diagrama Hierárquico Principal da Especificação "S"

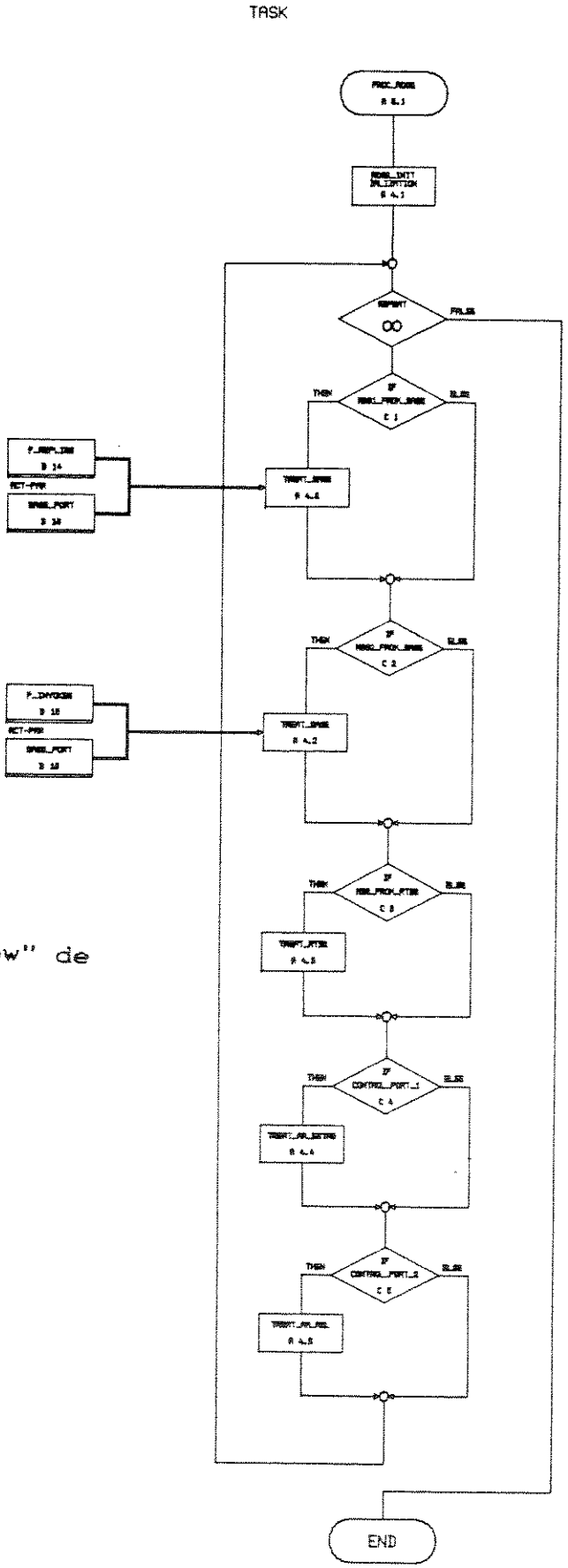


figura 3.5 - Diagrama "Dataflow" de PROC_ROSE (com RTSE na AE).

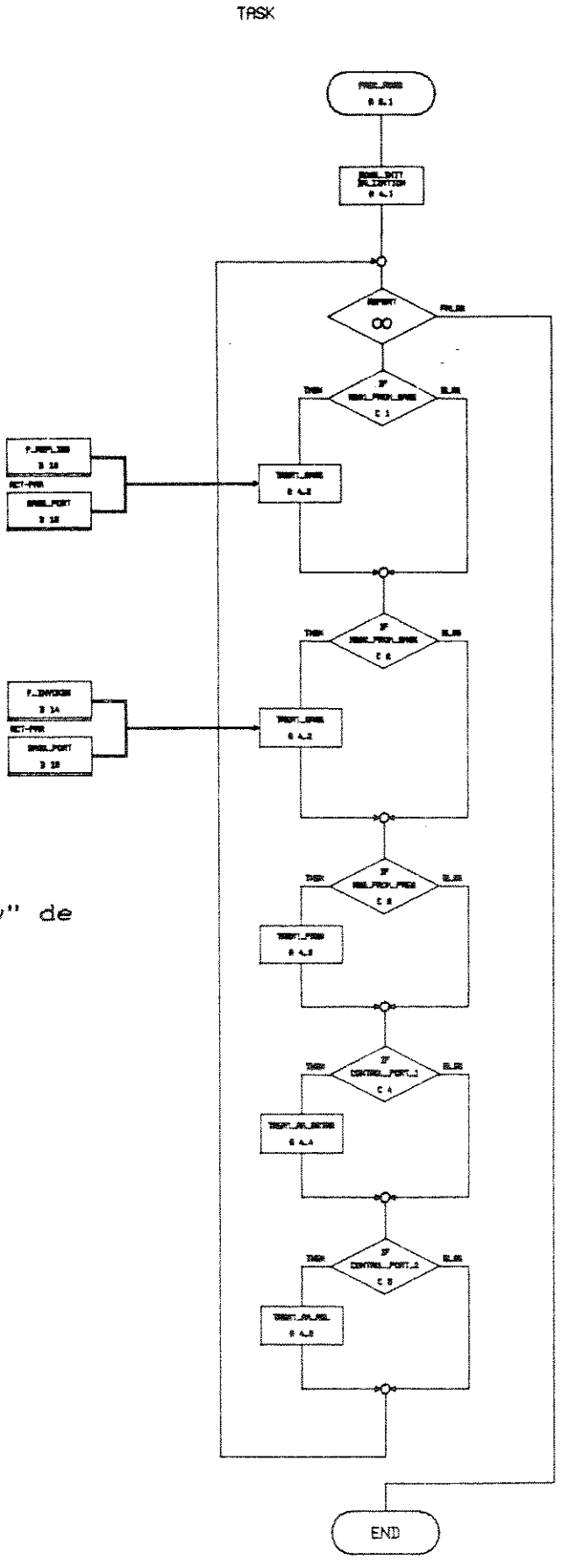


figura 3.6 - Diagrama "Dataflow" de PROC_ROSE (sem RTSE na AE).

No caso de MSG1_FROM_SASE ou MSG2_FROM_SASE serem verdadeiras, a ação TREAT_SASE é executada para receber uma mensagem, "reply" ou "invoke", do usuário SASE. Sua função é tratar as primitivas recebidas na interface da seguinte maneira:

1) Mapear a primitiva recebida na mensagem em um dos eventos de entrada conhecidos;
2) Determinar o estado atual da máquina de protocolos ROPM;
3) Dispondo das duas informações acima, executar uma ação que corresponda ao cruzamento destas informações na tabela de eventos/estados especificada em [17] e [18].

A ação TREAT_SASE é a formalização do Receptor de Eventos do Usuário no modelo criado para o ROSE, mostrado na figura 3.2.

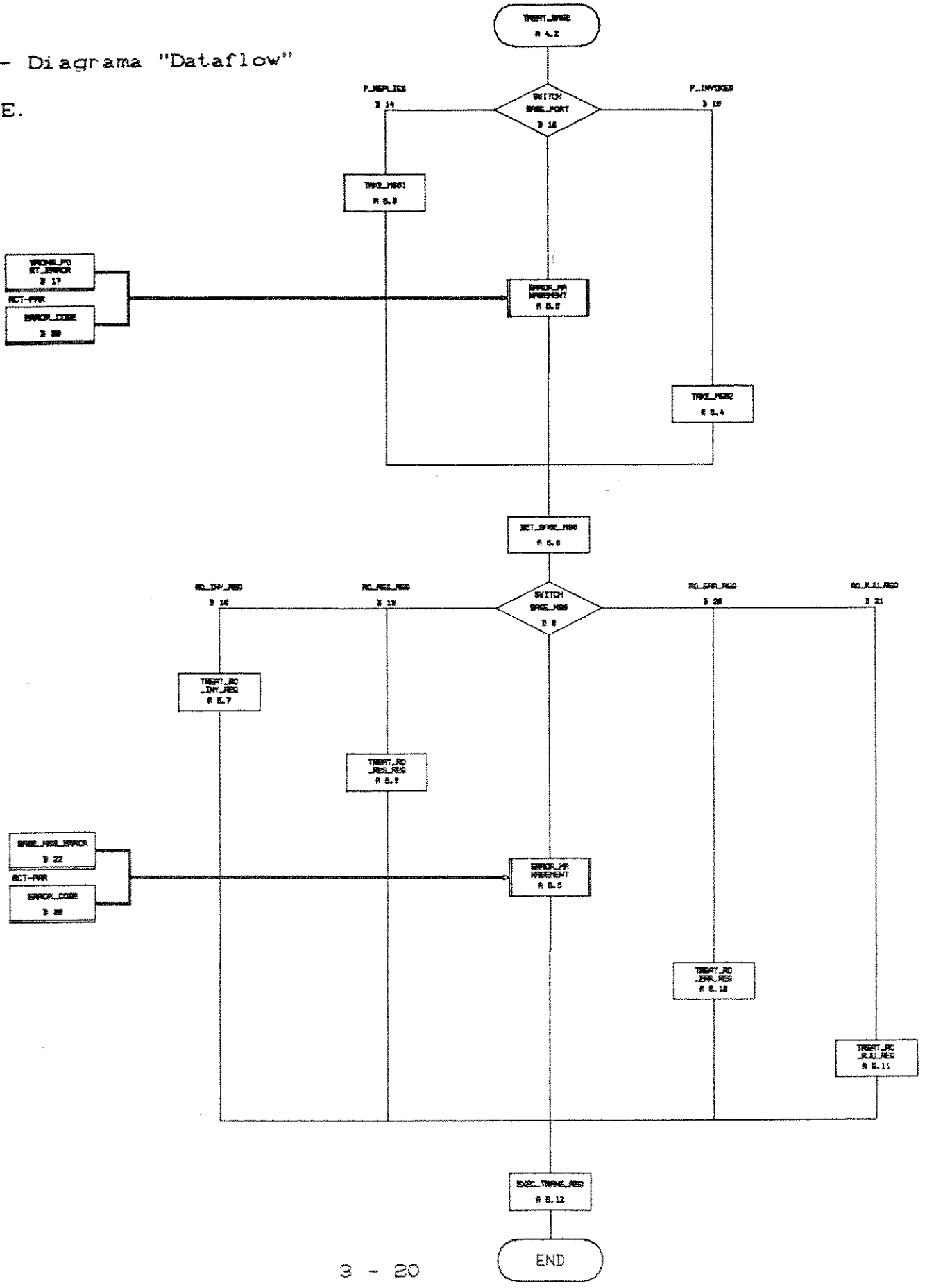
Caso a segunda condição MSG_FROM_RTSE seja verdadeira, TREAT_RTSE ou TREAT_PRES serão executadas. A operação destas ações é análoga à de TREAT_SASE, mudando apenas as primitivas recebidas, interface considerada e eventos gerados. As duas, cada uma em seu contexto, são a formalização do Receptor de Eventos do Provedor no modelo da figura 3.2. As Figuras 3.7 e 3.8 mostram diagramas "dataflow" da TREAT_SASE e TREAT_RTSE.

As ações componentes de TREAT_SASE são muito semelhantes, funcionalmente falando, pois todos eventos associados são primitivas de solicitação de transferência de APDU's. Logo, todos incluem uma ação que forma sua APDU específica, uma que executa o pedido de transferência para a ROPM_TR e uma que mantém o estado STA02 (associado) para a ROPM. Como o pedido de transferência para a ROPM-TR é comum a todos os componentes de TREAT_SASE, este procedimento é colocado ao final da ação. Para dar uma idéia melhor de como são os elementos da ação que trata eventos do usuário tem-se a seguir o diagrama de fluxo da ação TREAT_RO_INV_RFQ mostrado na Figura 3.9.

O pedido de transferência para a ROPM_TR é feito pela ação EXEC_TRANS_REQ, figura 3.10, que representa a passagem de uma

`igura 3.7 - Diagrama "Dataflow"

de TREAT_SASE.



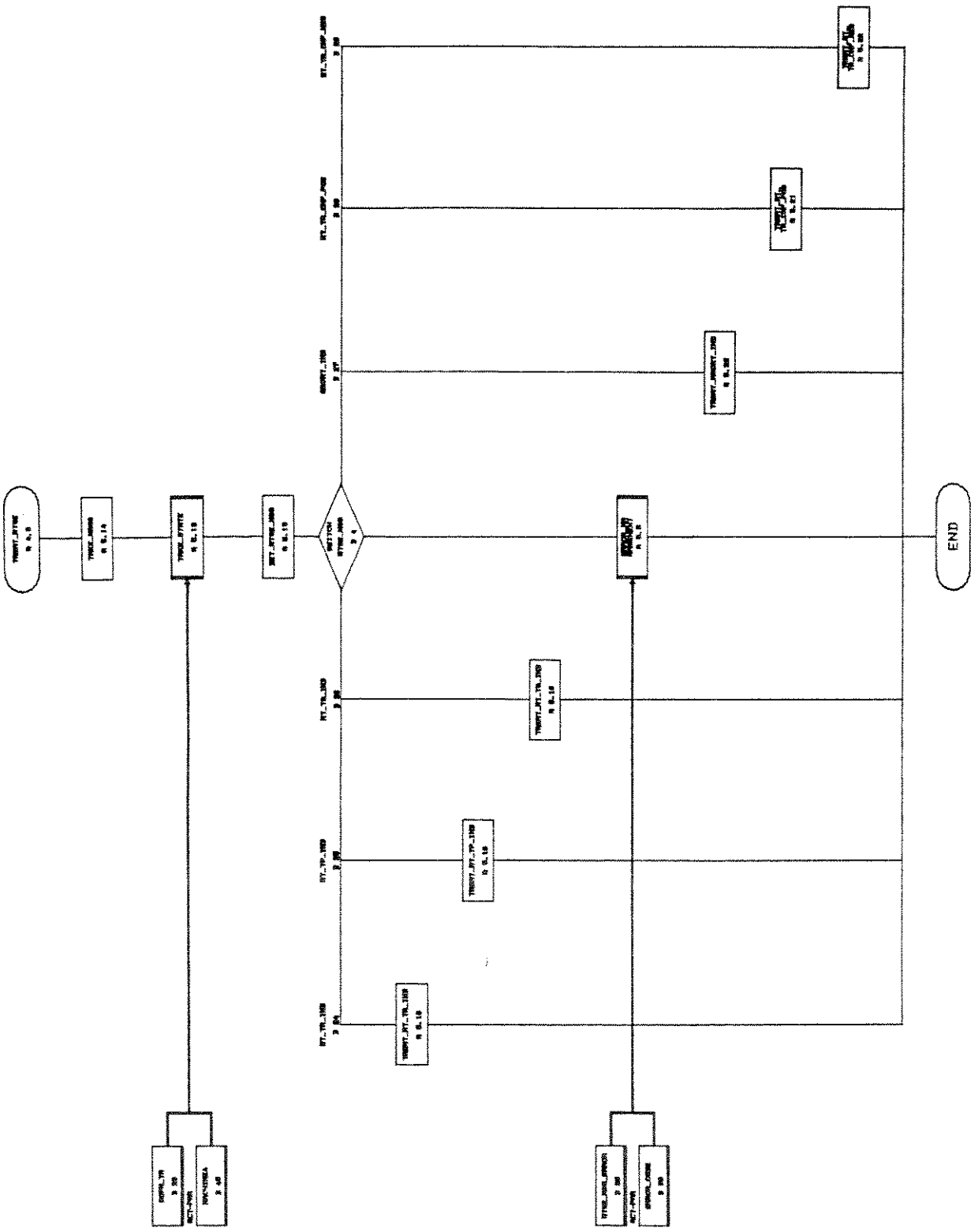


figura 3.8 - Diagrama "Dataflow" de TREAT_RTSE.

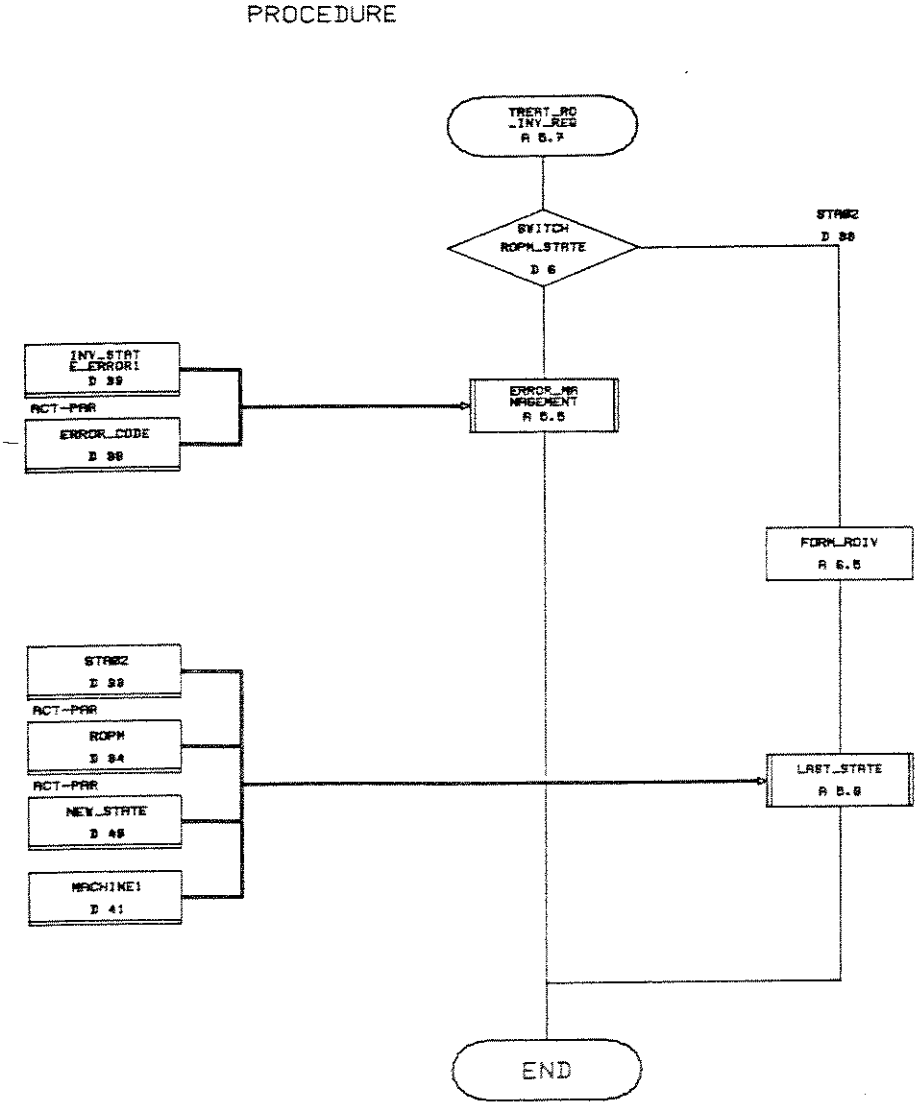
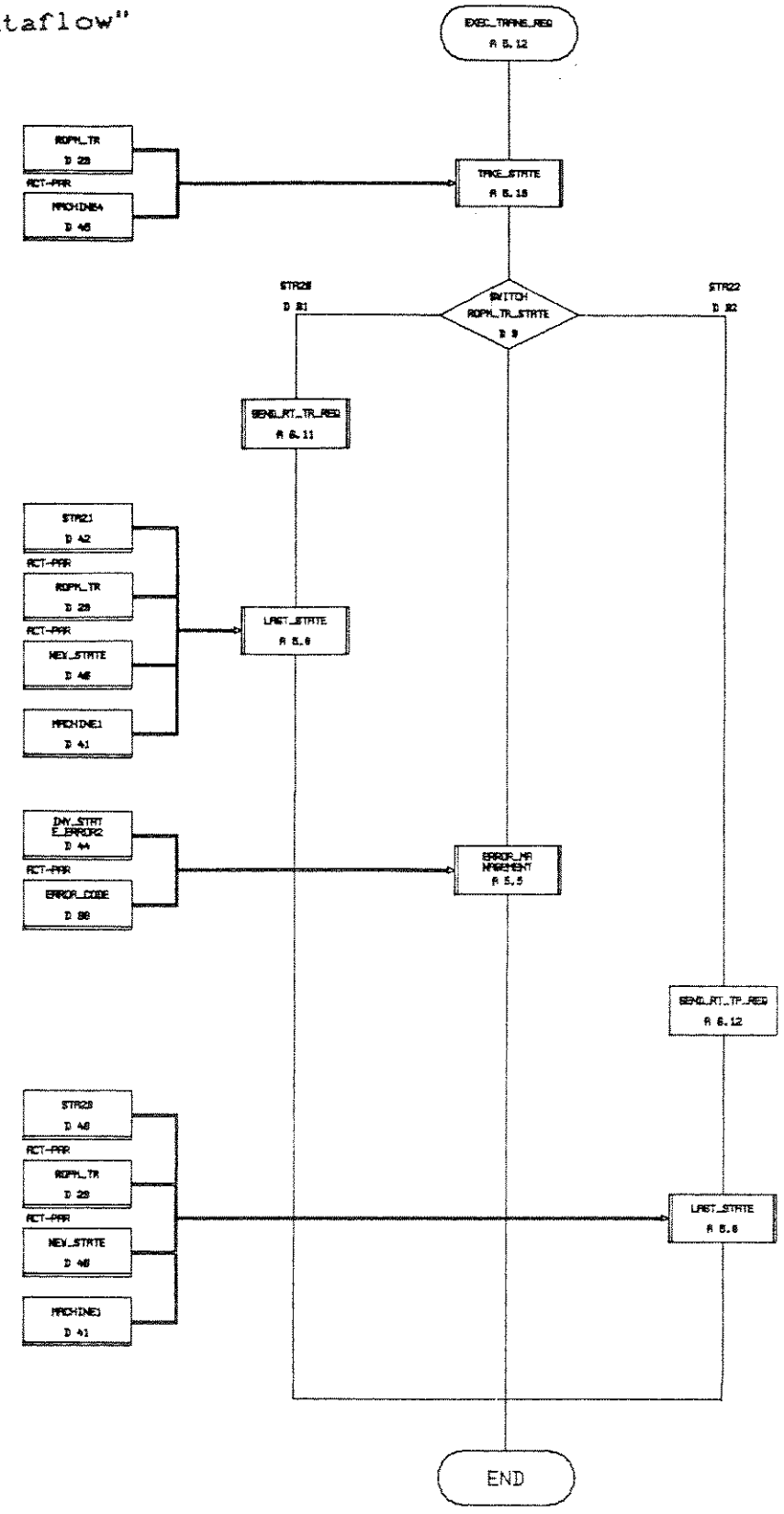


figura 3.9 - Diagrama "Dataflow" de TREAT_RO_INV_REQ.

figura 3.10 - Diagrama "Dataflow" de EXEC_TRANS_REQ.



primitiva de transferência interna da ROPM para a ROPM_TR. Juntamente com mais três outras ações compõe o módulo Gerador de Eventos Internos do modelo adotado.

As ações que tratam os eventos provenientes do provedor são bastante heterogêneas, se comparadas com as provenientes do usuário. As ações que formam TREAT_RTSE são apresentadas a seguir, juntamente com um gráfico tipo "dataflow" e breve explicação.

TREAT_RT_TR_IND

Ao chegar uma primitiva RT_TR_IND , esta ação deve preencher uma variável criada para a comunicação e chamar a ação que faz o transporte interno de uma APDU para a ROPM: SEND_RT_TR_IND. Implementa o REQUIREMENT 4(3) parcialmente. Ver figura 3.11.

TREAT_RT_TR_CNF_POS

Só é disparada quando um evento RT_TR_CNF_POS ocorre no estado STA22; caso contrário, uma geração de erro é feita. Consiste simplesmente em mudar o estado de ROPM_TR de STA22 para STA20, indicando que uma transferência foi bem sucedida. Atende ao REQUIREMENT 4(3) parcialmente.. Ver figura 3.12.

TREAT_RT_TR_CNF_NEG

É muito semelhante à anterior, mas ao contrário, indica que a transferência não foi confirmada e dispara uma ação que indica isto ao usuário. Obedece ao REQUIREMENT 4(3) parcialmente. Ver figura 3.13.

TREAT_RT_TP_IND

Só é executada quando um TOKEN de transmissao está no sistema local (STA20 e STA21), caso contrário, implica em erro de ocorrência inválida. se não houver transferência a fazer, ou em an-

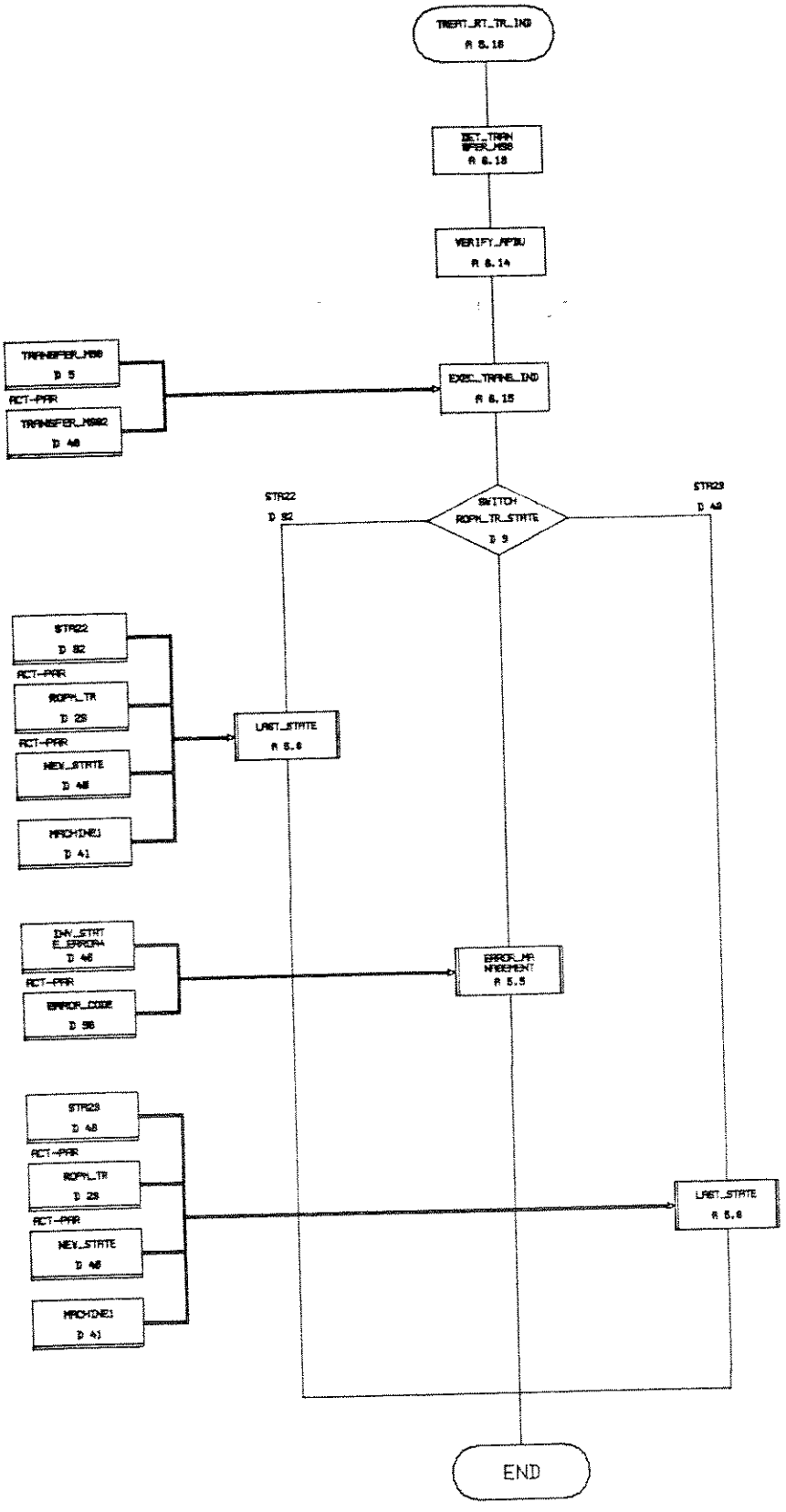


figura 3.11 - Diagrama "Dataflow" de TREAT_RT_TR_IND.

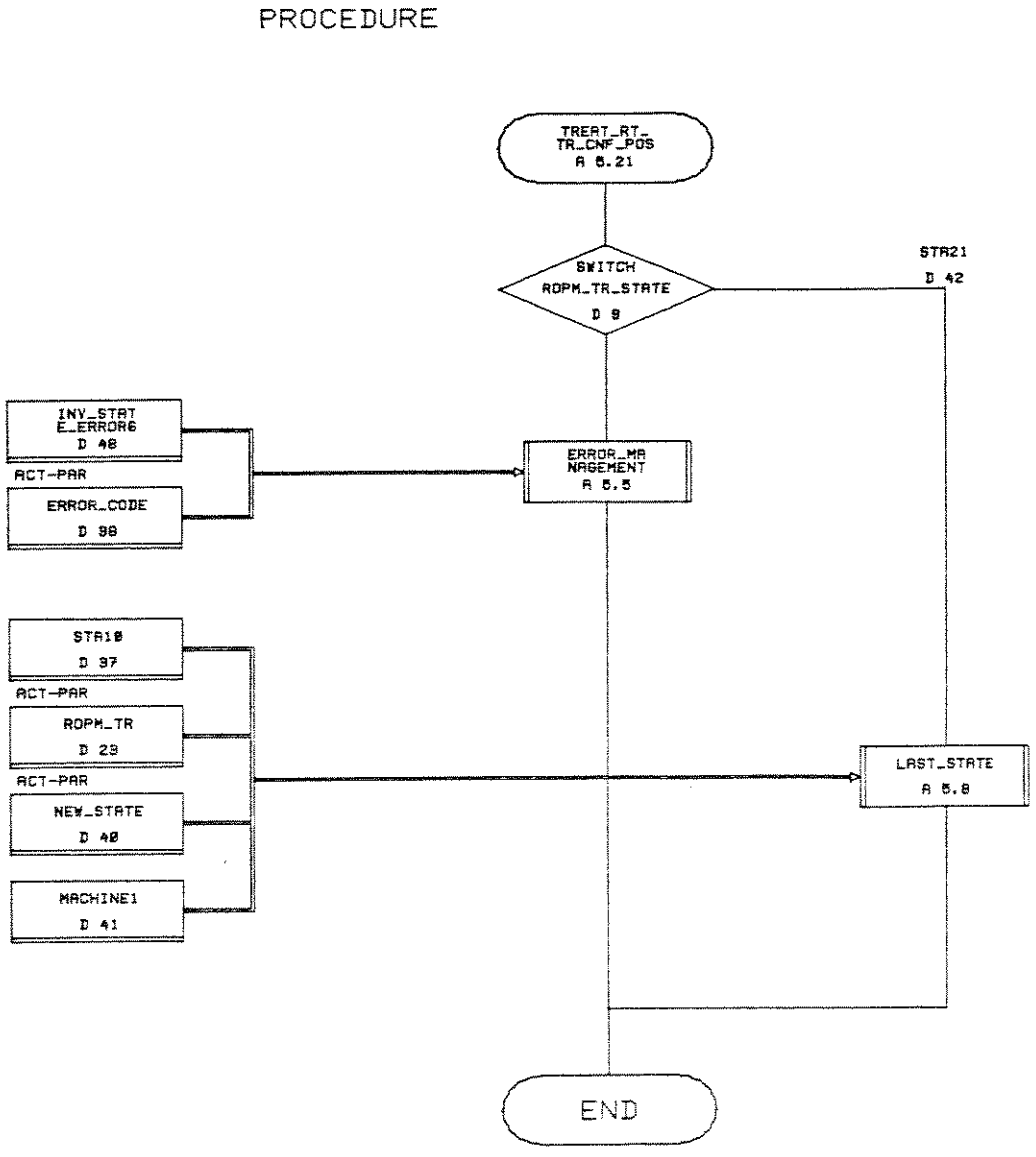


figura 3.12 - Diagrama "Dataflow" de TREAT_RT_TR_CNF_POS.

damento, um TOKEN é passado para o sistema remoto através do envio da primitiva RT_TG_REQ. Implementa REQUIREMENT 4(3) parcialmente. Ver figura 3.14.

TREAT_RT_TG_IND

Quando disparada indica a entrega do TOKEN ao sistema local. A máquina de estados ROPM_TR deverá estar em STA22 ou STA23. Caso haja uma mensagem a transferir, é disparada a ação SEND_RT_TR_REQ e depois muda pra o estado STA21. Se não houver mensagens a transmitir, a mudança é para o estado STA20. Implementa o REQUIREMENT 4(3) parcialmente. Ver figura 3.15.

TREAT_ABORT_IND

Implementa o tratamento a uma mensagem de aborto do provedor de serviços. Se houver transferências pendentes é gerada um ação SEND_RO_RJP_IND ao usuário, senão; se o estado for STA20 ou STA21, é executada a ação de transferência interna SEND_AA_AB_IND. Implementa REQUIREMENT 5(3) parcialmente. Ver figura 3.16.

No caso do contexto que não engloba o RTSE, como foi mostrado, a camada de Apresentação é o provedor do ROSE. Na implementação é representada pela ação TREAT_PRES. TREAT_PRES é formada apenas por duas ações: TREAT_P_DATA_IND, semelhante à TREAT_RT_TR_IND, e TREAT_ABORT_IND. Ver figura 3.17.

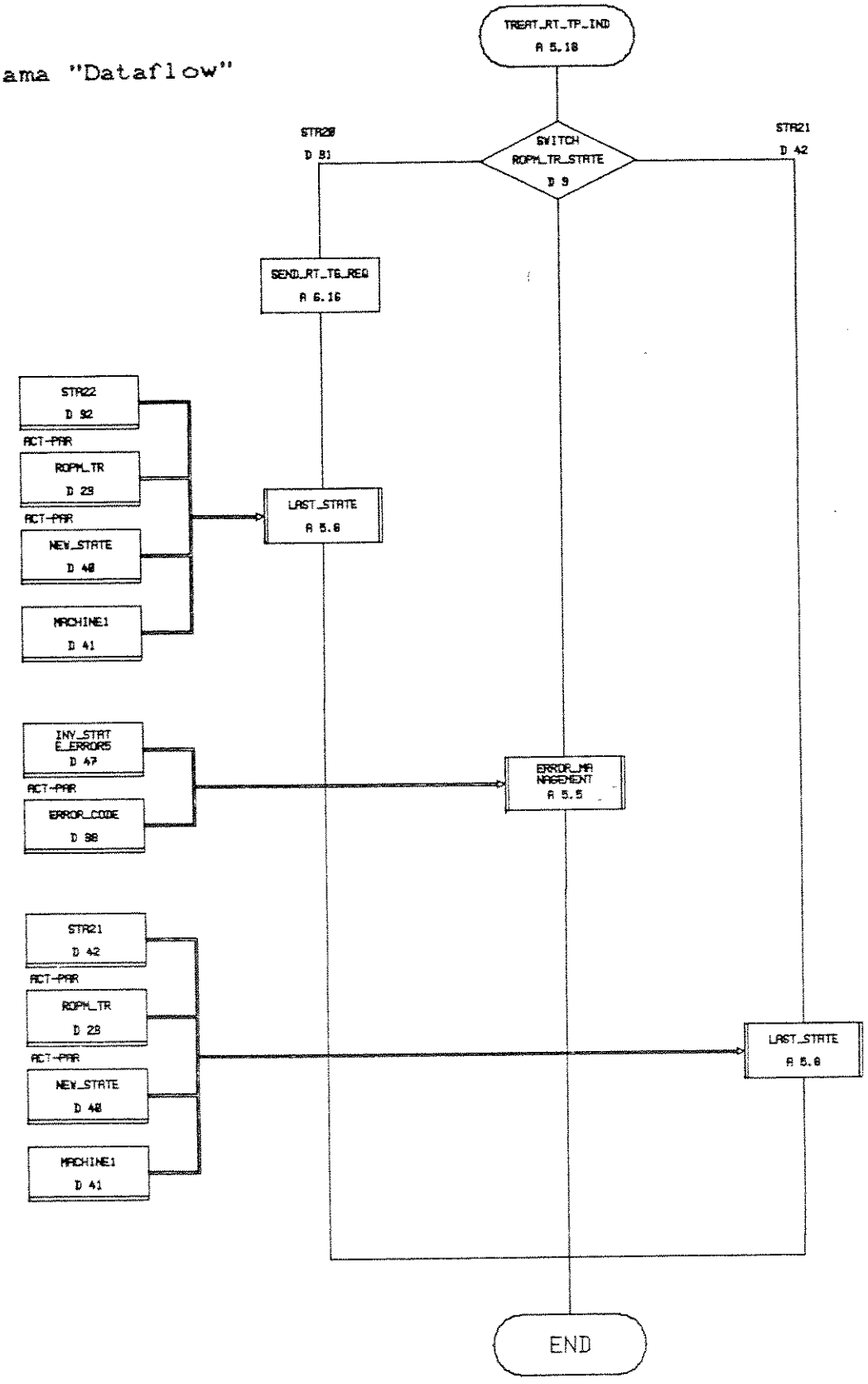
GERENCIAMENTO DE ERROS

Ao longo de toda a especificação, o uso da ação ERROR_MANAGEMENT foi constante. Sua função é imprimir uma mensagem de erro juntamente com o código de um erro detectado. Os possíveis erros são:

1) SASE_MSG_ERROR (0)
   Ocorre quando o evento recebido do SASE não é uma das seguintes:

figura 3.14 - Diagrama "Dataflow"

de TREAT_RT_TP_IND.



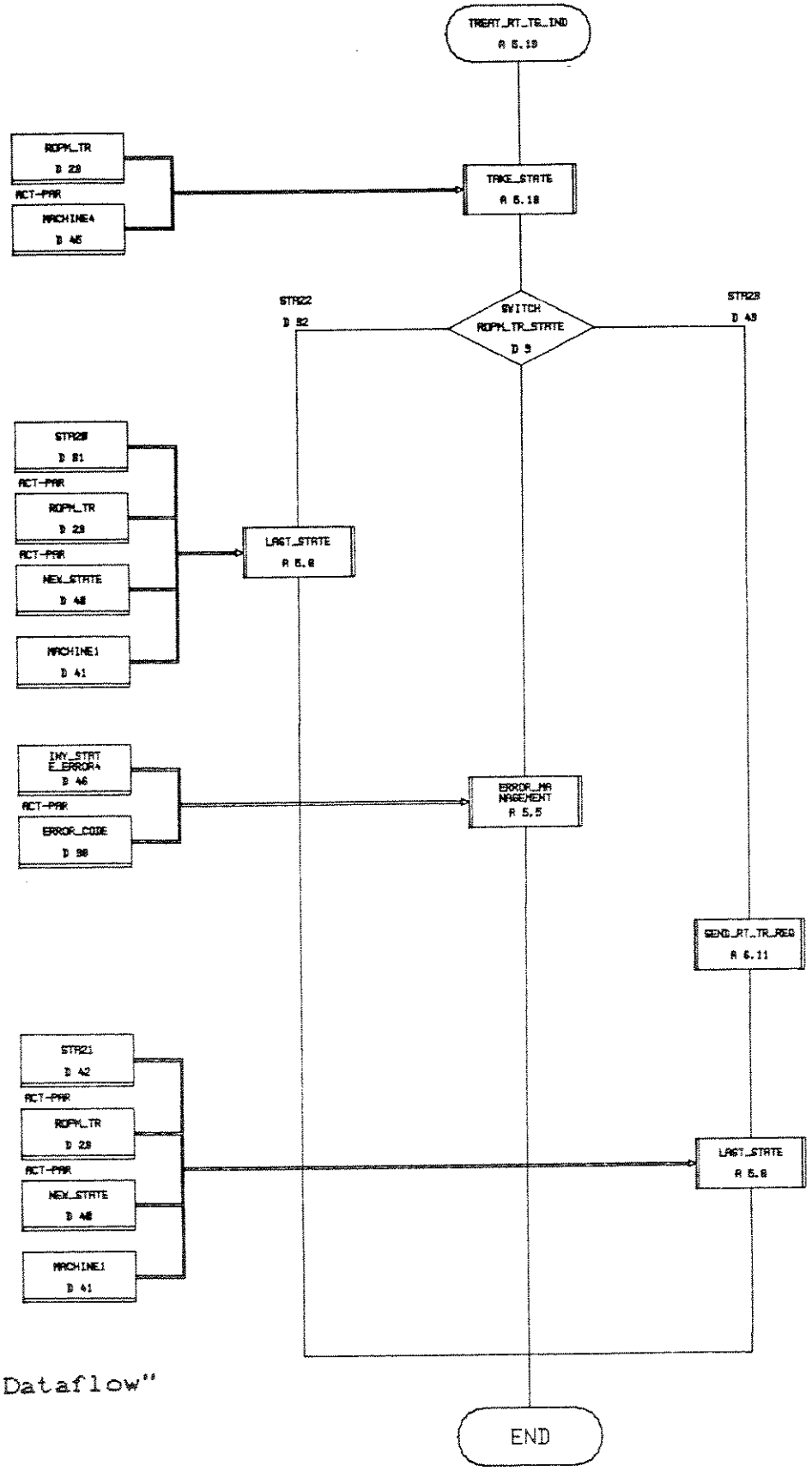


figura 3.15 - Diagrama "Dataflow"

e TREAT_RT_TG_IND.

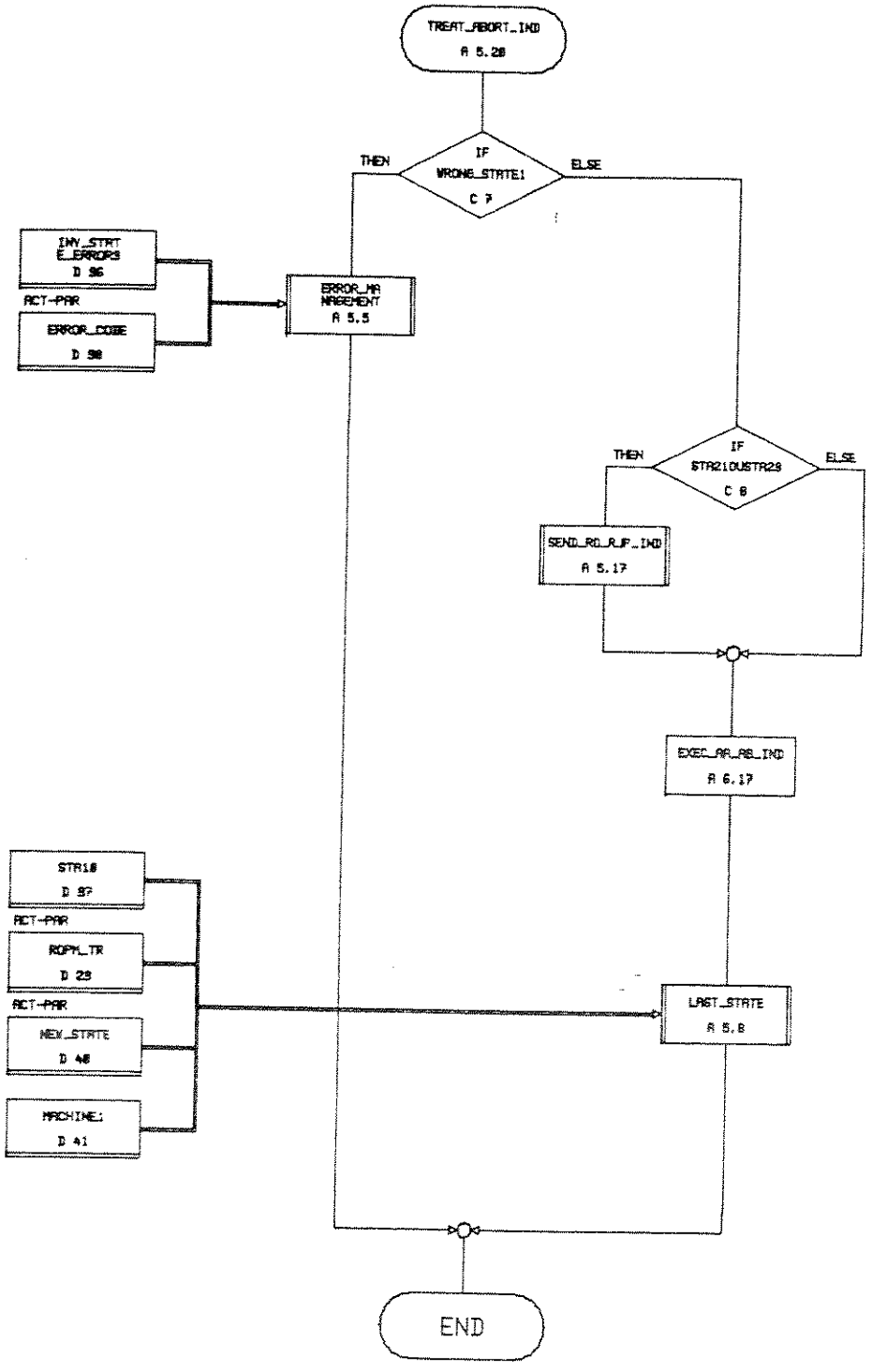


figura 3.16 - Diagrama "Dataflow" de TREAT_ABORT_IND.

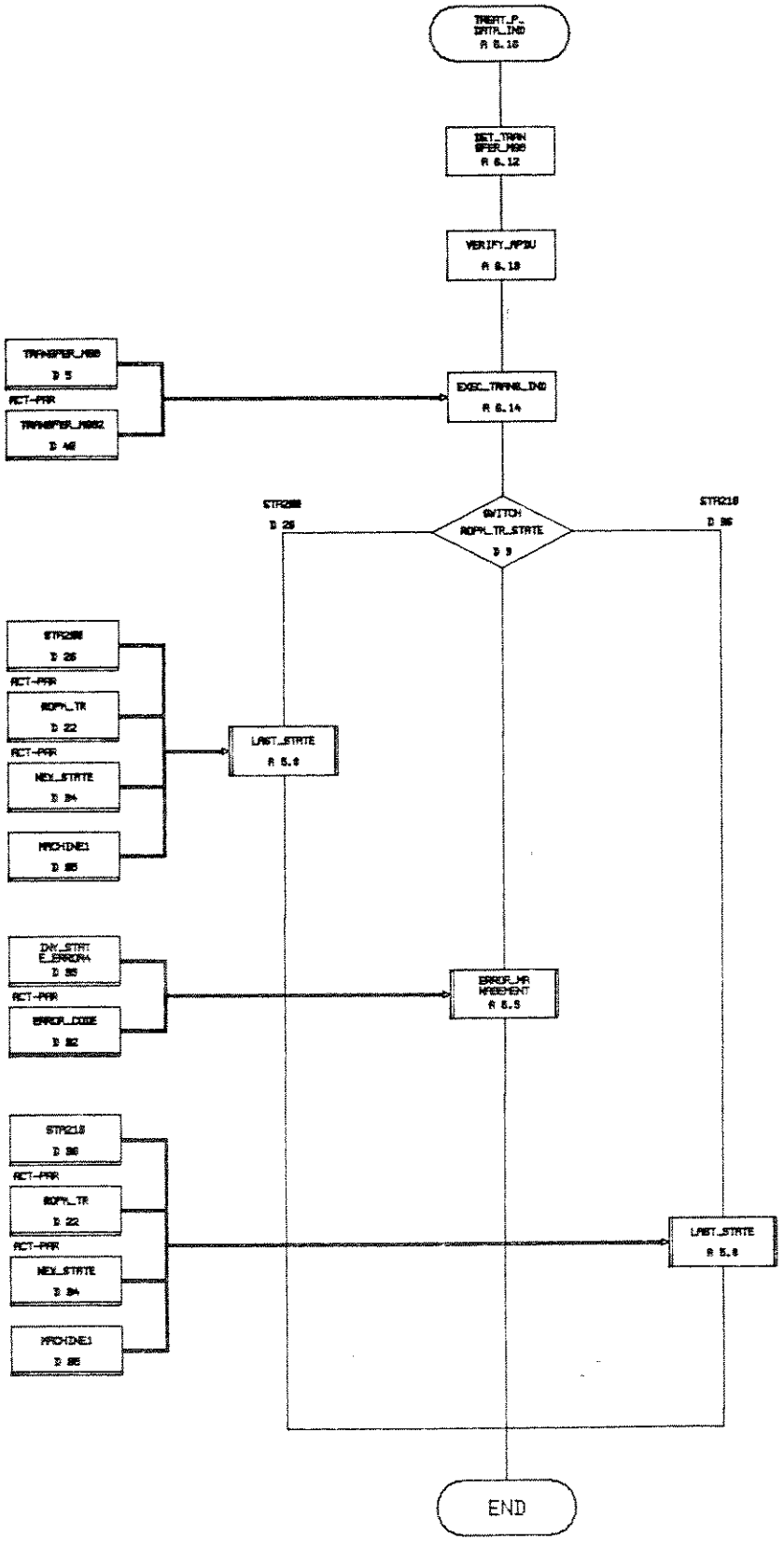


figura 3.17 - Diagrama "Dataflow" de TREAT_P_DATA_IND.

. RO_INV_REQ . RO_RES_REQ
. RO_ERR_REQ . RO_RJU_REQ

2) RTSE_MSG_ERROR ou
PRES_MSG_ERROR (1)

Ocorre quando o evento recebido de provedor não é uma das seguintes:

CONTEXTO COM RTSE	CONTEXTO SEM RTSE
. RT_TR_IND	. P_DATA_IND
. RT_TR_CNF_POS	
. RT_TR_CNF_NEG	
. RT_TP_IND	
. RT_TG_IND	
. ABORT_IND	

3) TRANSFER_MSG_ERROR (4)

Ocorre quando a APDU transportada na primitiva interna TRANS_IND é diferente de:

. ROIV . RORS . ROER . RORJU e . RORJP

4) INV_STATE_1 (5)

Ocorre quando o estado da máquina de protocolos ROPM é diferente de STA02.

5) INV_STATE_2 (6)

Ocorre quando o estado da máquina de protocolo ROPM_TR é diferente de :

CONTEXTO COM RTSE	CONTEXTO SEM RTSE
STA20 e STA22.	STA200 e STA210

6) INV_STATE_3 (7)

Ocorre quando o estado da máquina de protocolo ROPM_TR é diferente de :

CONTEXTO COM RTSE	CONTEXTO SEM RTSE
STA20, STA21, STA22 e STA23	STA200 e STA210

7) INV_STATE_4 (8)

Ocorre quando o estado da máquina de protocolo ROPM_TR é diferente de :

CONTEXTO COM RTSE	CONTEXTO SEM RTSE
STA23 e STA22	STA200 e STA210

8) INV_STATE_5 (9)

Ocorre quando o estado da máquina de protocolo ROPM_TR é diferente de :

CONTEXTO COM RTSE	CONTEXTO SEM RTSE
STA20 e STA21	STA200 e STA210

9) INV_STATE_6 (10)

Ocorre quando o estado da máquina de protocolo ROPM_TR é diferente de STA21.

GERENCIAMENTO DE ESTADOS/ASSOCIAÇÕES

O gerenciamento de estados/associações tem como base a estrutura de dados do tipo tabela que está associada à variável STATE_TAB. Esta variável implementa o bloco da figura 3.2 correspondente à Tabela de Estados e Associações. A figura a seguir mostra sua forma.

MÁQUINA	ROPM	ROPM_TR
ASSOCIAÇÃO		
0		
1		
2	TAB_ELEMENT	
i		
MAX_CONTEXT		

tabela 3.1 - Esboço da tabela de Estados/Conexões

A interação dos demais blocos do módulo com a tabela de controle é feita por meio das ações :

a) INITIATE_STATE_TAB - Inicializa a tabela colocando em cada entrada o indicador de nenhuma associação, NONE, e o estado inicial de ROPM e ROPM_TR em cada associação.

b) LAST_STATE - Tendo a máquina e novo estado como argumento, procura a posição da associação atual na tabela, e muda o valor do estado encontrado para o valor do argumento novo estado.

c) TREAT_AA_ESTAB e TREAT_AA_REL - Também alteram STATE_TAB da forma já discutida anteriormente, ou seja, acrescentam ou liberam uma linha da tabela, respectivamente, para estabelecer ou liberar uma associação.

TREAT_AA_ESTAB

Tem como função sinalizar para a máquina de protocolo (ROPM e ROPM_TR) o estabelecimento de uma associação. Sua ação limita-se a alterar uma das linhas da tabela de estados/conexão para ROPM e para a ROPM_TR. Implementa os REQUIREMENT's 3(1), 1(3) parcialmente e 4(3) também parcialmente. Veja figura 3.18.

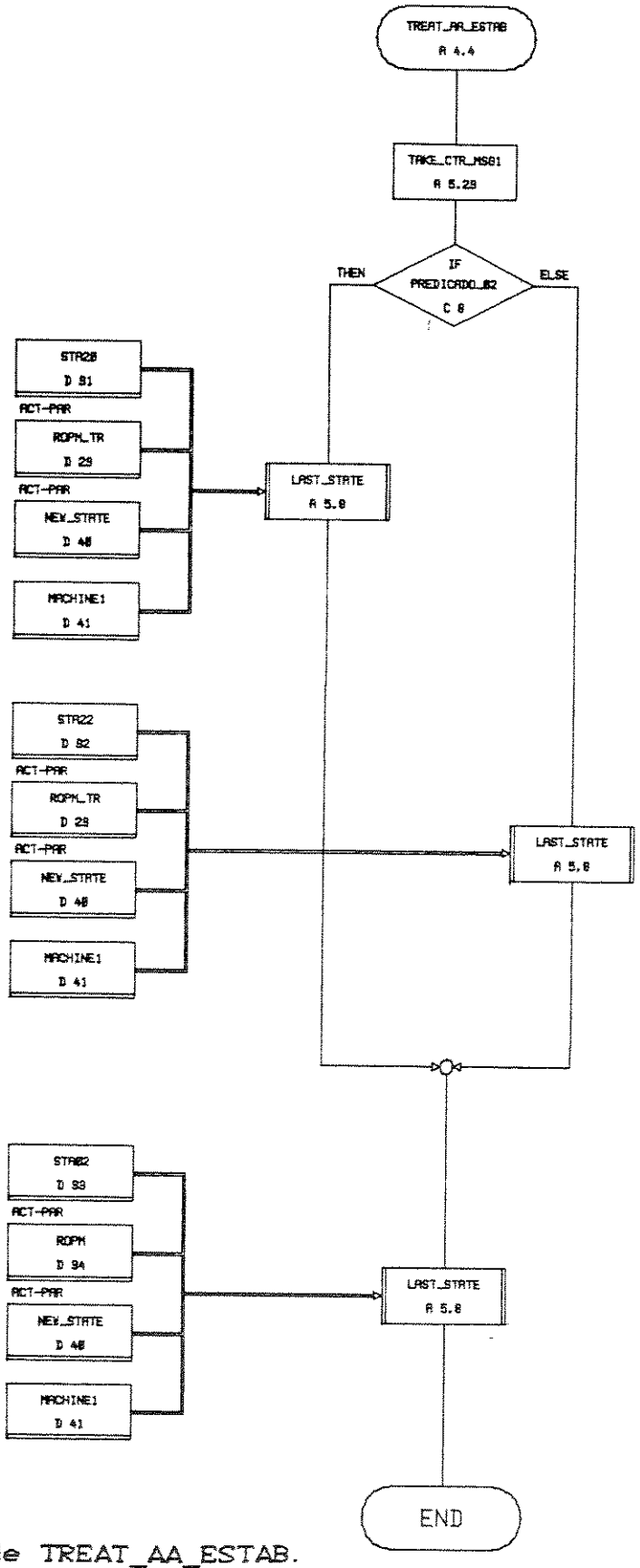


figura 3.18 - Diagrama "Dataflow" de TREAT_AA_ESTAB.

TREAT_AA_REL

Tem como função indicar a liberação de uma associação. Sua ação também limita-se à alteração de uma linha da tabela de estados/conexão mudando o estado de ROPM e ROPM_TR para dissociado, respectivamente, STA01 e STA10. Implementa os REQUIREMENT's 1(3) e 5(3), ambos parcialmente. Veja figura 3.19.

"LOCAL IMPLEMENTATION MATTERS"

As questões de implementação encontradas no documento são:

. Comunicação com protocolos vizinhos;
. Implementação de controle das prioridades nos serviços do ROSE;
. Determinação de valores de constantes de temporização e número de rejeição;
. Indicação de que uma associação foi estabelecida/liberada.

A comunicação com protocolos vizinhos é uma questão resolvida de antemão no início do projeto SISDI_MAP, ou seja, o núcleo dedicado se encarrega de criar portas específicas a cada processo que, mesmo permitindo entrada e saída numa mesma porta, são consideradas apenas de saída, ou apenas de entrada, para cada protocolo vizinho considerado.

No sistema de comunicação apresentado, o envio e retirada de uma mensagem para/da fila são feitos pelas rotinas definidas no núcleo: envia_mens e espera_mens, respectivamente. Na especificação, correspondem às seguintes ações:

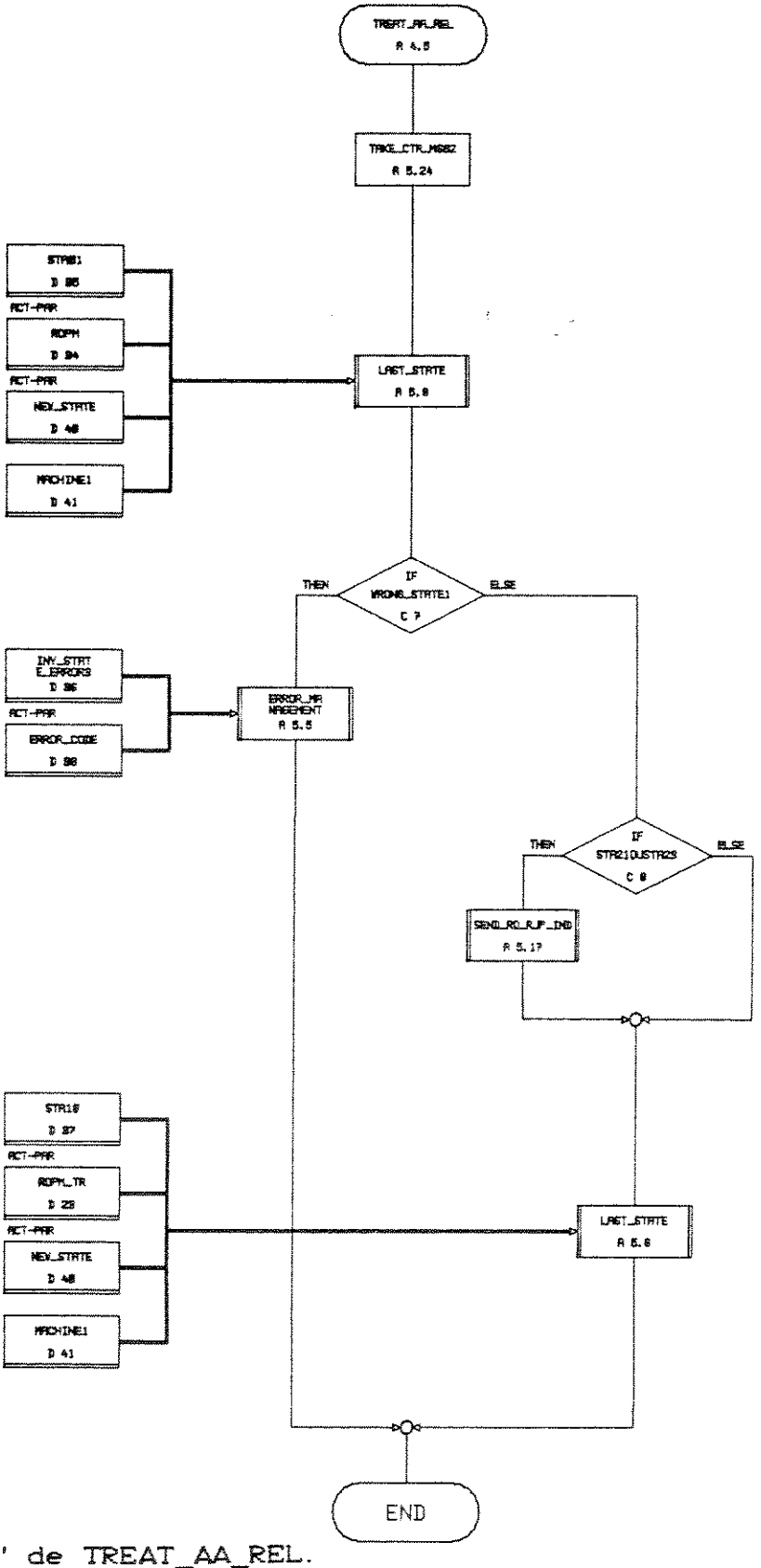


figura 3.19 - Diagrama "Dataflow" de TREAT_AA_REL.

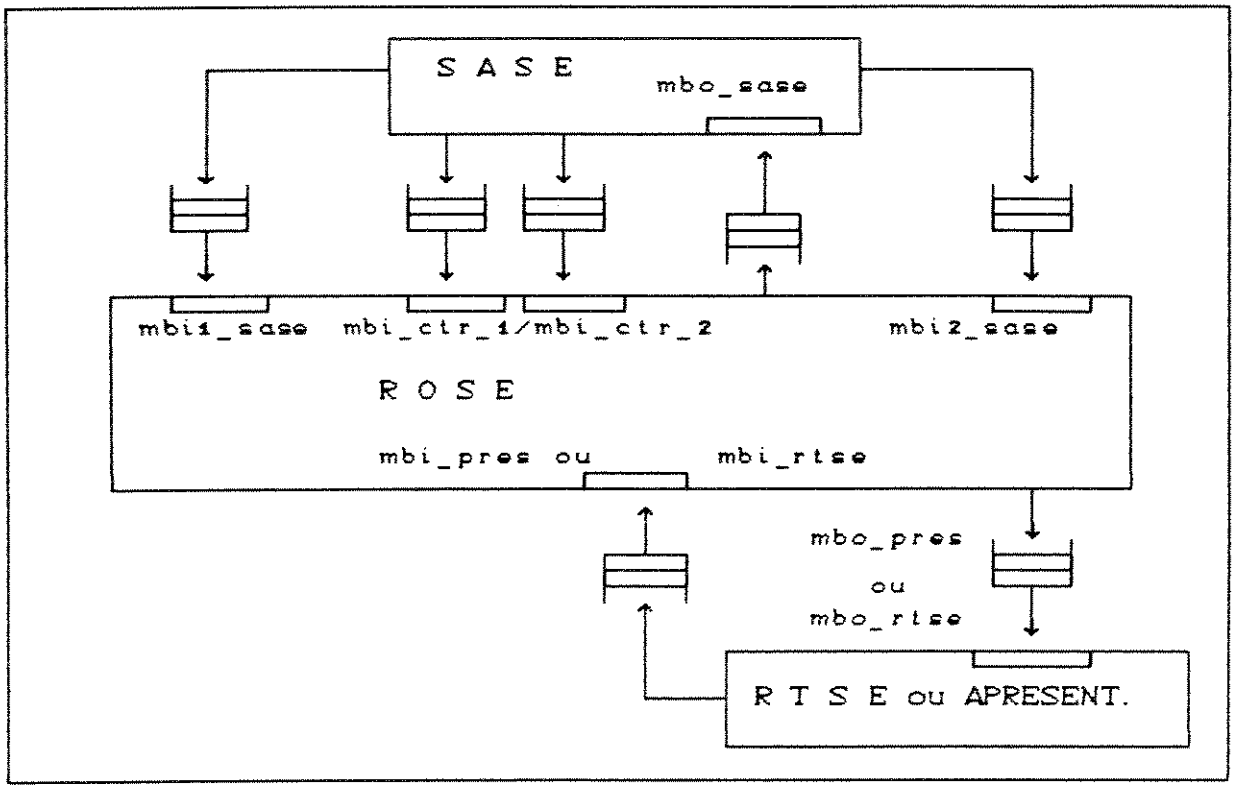


Figura 3.20 - Esquema de Comunicação Externa do ROSE

1) espera_mens	<=>	MSG1_FROM_SASE	MSG2_FROM_SASE
		MSG_FROM_RTSE	
		MSG_FROM_ROPM	
		MSG_FROM_ROPM_TR	
2) envia_mens	<=>	SEND_RO_INV_IND	
		SEND_RO_RES_IND	
		SEND_RO_ERR_IND	
		...	
		SEND_RT_TR_REQ	
		SEND_RT_TG_REQ	
		...	
		ROPM_RECEIVE	
		ROPM_TR_RECEIV	

As primitivas de serviço do ROSE têm um parâmetro chamado *prioridade* que define a ordem de atendimento das primitivas oriundas do SASE. Isto ocorre porque o protocolo privilegia primitivas

que sejam respostas a invocações, em detrimento de invocações que ainda não foram feitas ao sistema remoto. A princípio, era de se esperar que o próprio núcleo tivesse um recurso de priorização de mensagens nas filas de comunicação que gerencia. No entanto, por ser bastante simplificado, o núcleo não dispõe deste controle.

A solução encontrada foi associar portas diferentes a primitivas de prioridades diferentes, de forma que o próprio protocolo possa fazer um controle que privilegia a porta de maior prioridade, ou seja, às primitivas correspondentes às respostas de invocações já feitas.

As constantes existentes na especificação fornecida pela ISO, foram calculadas arbitráriamente. A razão disto está no fato dos processos não estarem devidamente integrados, pois de outra forma fica difícil fazer uma definição de um valor ótimo para as mesmas.

Na especificação fornecida pela ISO no documento ISO 9072/2, existem dois eventos chamados AA_ESTAB e AA_REL que são descritos conforme a tabela abaixo:

	CONTEXTO COM RTSE	CONTEXTO SEM RTSE
AA_ESTAB	RT_OPEN.resp + RT_OPEN.conf +	A_ASSOCIATE.resp + A_ASSOCIATE.conf +
AA_REL	RT_CLOSE.resp + RT_CLOSE.conf +	A_RELEASE.resp + A_RELEASE.conf +

tabela 3.2. Primitivas causadoras dos eventos AA_ESTAB/AA_REL

A tabela, aparentemente, sugere que o ROSE utilize os serviços de estabelecimento/liberação de conexão dos elementos de serviço: RTSE ou ACSE. No entanto, para que isto fosse possível uma das situações abaixo deveria ocorrer:

a) Existência no ROSE de serviços específicos que permitissem ao usuário SASE estabelecer/liberar uma associação, de forma que posteriormente o ROSE podesse mapeá-los para as primitivas RT_OPEN/CLOSE ou A_ASSOCIATE/RELEASE apropriadas;

b) Utilização do serviço RO_INVOKE para transportar as instâncias BIND/UNBIND e posteriormente mapeá-las para as primitivas RT_OPEN/CLOSE ou A_ASSOCIATE/RELEASE apropriadas ou;

c) Usar um artifício dependente das condições impostas na tabela acima e que respeitasse os mapeamentos e máquinas de estado especificados em ISO 9072/2 [15].

A opção c foi a escolhida. As razões para tal estão no fato de não existirem serviços específicos no ROSE para estabelecimento e liberação de associações e do documento ISO 9072/2 deixar bem claro que o serviço RO_INVOKE é mapeado apenas para os serviços RT_TRANSFER (RTSE) e P_DATA (APRESENTAÇÃO).

O artifício usado consiste simplesmente na sinalização pelo usuário da chegada de uma primitiva do tipo *confirm* positivo ou de um envio de uma primitiva do tipo *response* positivo. Esta sinalização é feita por meio do envio de uma mensagem de controle a uma das portas mbi_ctr_1 ou 2 . Veja a figura 3.21.

Um evento AA_ESTAB ou AA_REL será reconhecido se chegar alguma mensagem, originária de um SASE, nas portas mbi_ctr_1 ou mbi_ctr_2, respectivamente.

Esta mensagem de controle é simulada através do bloco de mensagens do protocolo, utilizando para isto apenas o campo do seu cabeçalho que contém a identificação de uma associação.

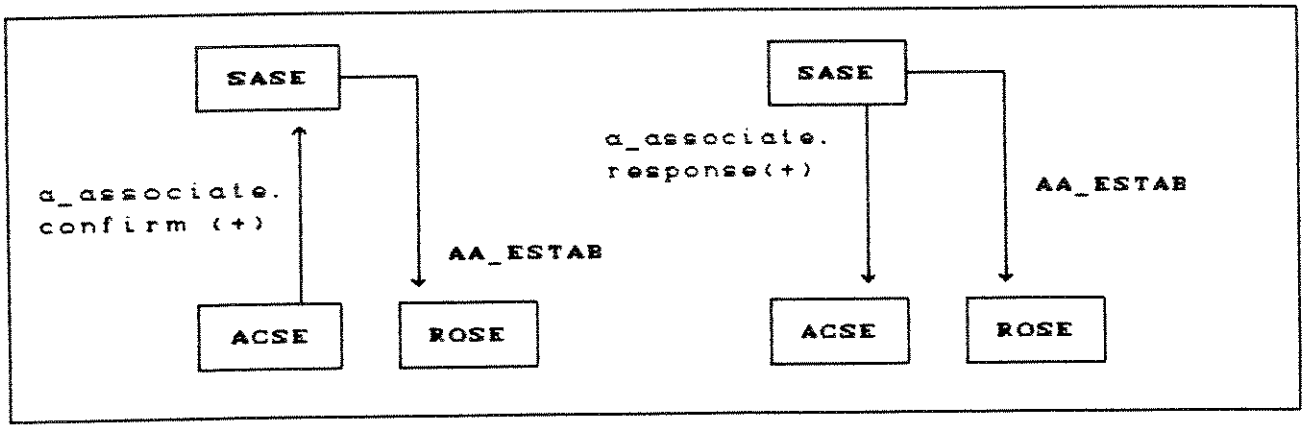


figura 3.21 - Implementação de AA_ESTAB em Contextos sem RTSE

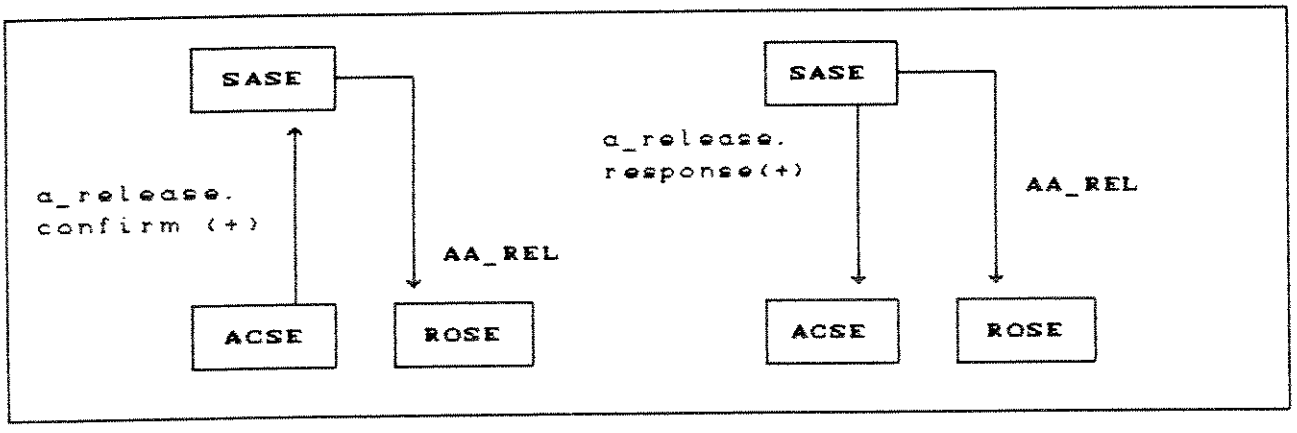


figura 3.21 - Implementação de AA_REL em Contextos sem RTSE

Algumas Estatísticas da Implementação

Terminada a especificação formal, usa-se o EPOS-A para fazer uma análise lógica e teste de conformidade com o que foi definido no EPOS-R. Os resultados são mostrados no APÊNDICE B.

A Especificação final que foi aprovada nas duas análises realizadas é formada de:

. **INCLUINDO RTSE** : 144 objetos; destes, 75 são ACTION, 56 são DATA, 12 são CONDITION e 1 é DEVICE.

. **SEM RTSE**: 129 objetos; destes, 68 são ACTION, 49 são DATA, 11 são CONDITION e 1 é DEVICE.

3.3. Geração Automática de Código

O COMPOSER é o módulo responsável pela geração automática de código. O EPOS dispõe de codificação para várias linguagens finais. Entre elas estão ADA, FORTRAN, PASCAL e C. Utilizamos o C pelas razões já expostas no início do capítulo.

A geração é feita somente a partir da especificação S. Isto ocorre através do mapeamento dos objetos da especificação em estruturas da linguagem, ou seja, alguns objetos são mapeados para dados e condições lógicas, e alguns o são para rotinas ou estruturas de controle.

A tarefa do COMPOSER é definida a partir da criação dos objetos no EPOS-S, pois é nesta etapa que se define qual o tipo de geração. Existem dois métodos:

a) Associar a cada objeto um segmento de programa no ítem <CODE-part>;
b) Associar código apenas aos objetos terminais da especificação.

O primeiro método tem as seguintes características:

- permite gerar programas mais otimizados e ao mesmo tempo usufruir da documentação e análise do EPOS;
- gera código para linguagens não disponíveis na versão do EPOS disponível.
- o projetista encarrega-se de verificar se a <CODE-part> com a <DECOMPOSITION-part> de cada objeto de projeto da especificação são equivalentes.

No segundo método, apenas os objetos terminais têm <CODE-part>, Desta forma, todo o fluxo de controle é gerado automaticamente pelo COMPOSER. Suas características são:

- não há necessidade de se testar o fluxo de controle;
- somente o código das ações terminais deve ficar sob responsabilidade do projetista;
- só permite linguagens da versão disponível;
- não há uma boa otimização do código final.

O método escolhido é o segundo pelo fato da linguagem C ser fornecida na versão 5.0.

A sequência de procedimentos para a obtenção do código final consiste em:

1º) escolher um dos objetos da especificação S como sendo o objeto inicial a partir do qual será criado um programa;

2º) aguardar que o COMPOSER realize as seguintes análises na especificação:

- checa se a especificação está completa. Ex.: verifica se os objetos mencionados foram definidos;
- checa se a especificação está consistente. Ex.: verifica se algum objeto foi definido mais de uma vez.
- checa se nenhuma exigência básica da linguagem escolhida foi desconsiderada na especificação.

3º) supondo que a especificação tenha sido aprovada nos testes, será feito, então, um mapeamento entre os objetos de projeto e as estruturas de um programa "C", conforme as regras seguintes:

a) Se deseja-se o programa principal do projeto, o objeto inicial tem que ser uma ACTION MODULE que contenha na sua <DECOMPOSITION-

part> uma ACTION TASK com o item TRIGGERED SYSTEMSTART definido. Caso contrário, será gerado apenas uma função comum.

b) Os objetos DATA são mapeados em variáveis ou constantes conforme especificado em [21];

c) Os objetos DATA TYPE são mapeados em estruturas de dados da linguagem "C" conforme [21];

d) CONDITION's são correspondentes às condições lógicas encontradas em um programa. Têm sua <CODE-part> integralmente substituida na posição que a condição terá no programa final;

e) Os objetos EVENT, INTERFACE, DEVICE não têm correspondentes diretos para a linguagem "C". (Logo, se a especificação analisada contiver algum deles, o COMPOSER, simplesmente ignora sua existência;

Apresenta-se a seguir um exemplo de mapeamento da ACTION TASK PROC_ROSE para código C:

```
ACTION TASK PROC_ROSE
DESCRIPTION :
PURPOSE :    "representa o programa principal do processo
             proc_ROSE".
CATEGORY : 'PROGRAMA-PRINCIPAL'.
DESCRIPTIONEND
DECOMPOSITION :
      ROSE_INITIALIZATION;
      REPEAT INFINITE

      IF MSG1_FROM_SASE THEN TREAT_SASE USING (P_REPLIES) FI;
      IF MSG2_FROM_SASE THEN TREAT_SASE USING (P_INVOKES) FI;
      IF MSG_FROM_RTSE  THEN TREAT_RTSE FI;
      IF CONTROL_PORT_1 THEN TREAT_AA_ESTAB FI;
      IF CONTROL_PORT_2 THEN TREAT_AA_REL FI

ACTIONEND
```

listagem 3.1 - Definição da ação TASK PROC_ROSE

A partir de uma especificação, como a da listagem 3.1, o COMPOSER faz um mapeamento de objetos para estruturas da linguagem final, no caso C, e gera a sequência de código equivalente. er listagem seguinte:

```
PROC_ROSE()
{ /*------------ ATRIBUIÇÕES ---------------*/
  mbi1_sase = ingate1_sase;
  mbi2_sase = ingate2_sase;
  mbi_rtse  = ingate_rtse;
  mbo_sase  = outgate_sase;
  mbo_rtse  = outgate_rtse;
  mbo_sase  = outgate_sase;
  mbo_rtse  = outgate_rtse;

  INITIATE_STATE_TAB();
  for ( ; ; )
  { if (espera_mens(mbi1_sase,WAIT_TIME,VERIFICA,block)==
        OPERACAO_OK)
       { TREAT_SASE ( P_REPLIES ) }

    if ((!(espera_mens(mbi1_sase,WAIT_TIME,VERIFICA,block)==
       OPERACAO_OK) && (espera_mens(mbi2_sase,WAIT_TIME,
       VERIFICA,block)==OPERACAO_OK))
       { TREAT_SASE ( P_INVOKES ) }

    if (espera_mens(mbi_rtse,WAIT_TIME,VERIFICA,block)==
       OPERACAO_OK)
       { TREAT_RTSE () }

    if (espera_mens(mbi_ctr_1,WAIT_TIME,VERIFICA,block)==
       OPERACAO_OK)
       { TREAT_AA_ESTAB () }

    if (espera_mens(mbi_ctr_2,WAIT_TIME,VERIFICA,block)==
       OPERACAO_OK)
       { TREAT_AA_REL () }

  }

} /* PROC_ROSE */
```

listagem 3.2 - Código Gerado a partir do objeto PROC_ROSE

No código final, algumas linhas foram geradas automáticamente e algumas manualmente. Isto implica na verdade em uma geração semi-automática, que tem uma performance maior quanto maior for a relação:

linhas obtidas automaticamente/linhas obtidas manualmente

O programa final, considerando o contexto com RTSE, tinha aproximadamente 600 linhas de código das quais as seguintes proporções foram verificadas:

ESTRUTURA	FORMA DE GERAÇÃO DAS LINHAS		
		Manual	Automática
	Atribuições	100.0 %	0 %
	IF-THEN-ELSE	45.4 %	54.6 %
	SWITCH-CASE	12.5 %	87.5 %
	TOTAL	32,4 %	67,6 %

tabela 3.2 - Percentuais de Codificação Automática/Manual com RTSE

Na implementação que não considera o elemento de serviço RTSE, os seguintes resultados foram calculados:

ESTRUTURA	FORMA DE GERAÇÃO DAS LINHAS		
		Manual	Automática
	Atribuições	100.0 %	0 %
	IF-THEN-ELSE	45.4 %	54.6 %
	SWITCH-CASE	12.2 %	87.8 %
	TOTAL	39,0 %	61,0 %

tabela 3.2 - Percentuais de Codificação Automática/Manual sem RTSE

Como pode-se ver, na implementação sem o RTSE o número de linhas codificadas manualmente aumentou, percentualmente falando, pelo fato de sua especificação apresentar uma tabela de eventos/estados mais simplificada; Por serem as tabelas de eventos/estados totalmente colocadas em especificação "S", ao diminuir sua complexidade diminui-se também a quantidade de código gerado automáticamente. Por serem praticamente equivalentes os segmentos de código manuais, aumenta-se consequentemente a proporção de codificação manual na segunda implementação.

4. Exemplos de Funcionamento do Processo ROSE

Neste capítulo, a implementação do ROSE que engloba o RTSE no contexto de aplicação é tomada como exemplo para que se faça uma descrição detalhada da interação entre os objetos da implementação durante o tratamento de um evento qualquer e da comunicação entre o processo que representa o ROSE (proc_ROSE) e os seus processos vizinhos. Estas duas interações são analisadas para os seguintes casos:

a) Utilização de um serviço do ROSE por um SASE e
b) Tratamento de uma mensagem proveniente da entidade remota via provedor de serviços.

Nas descrições apresenta-se como os eventos são identificados a partir da estrutura de comunicação existente, como é indicada a conexão a que se destinam, como são transferidas as mensagens entre os processos e que objetos de projetos foram usados em cada etapa.

Utilização de um Serviço do ROSE

Suponhamos que um SASE, especificado e implementado baseado na "RO-NOTATION", deseje realizar uma invocação de uma operação através do serviço "ro-invoke.request". Para tanto deverá seguir as seguintes etapas:

1º) Estabelecer uma associação através do elemento de serviço ACSE de forma que o processo ROSE possa construir sua tabela de conexões pelo tratamento dado ao evento AA_ESTAB;

2º) Preencher, no bloco de mensagens do protocolo, o cabeçalho e a área de dados do ROSE que corresponde aos parâmetros da primitiva "ro-invoke-request", conforme mostrado a seguir:

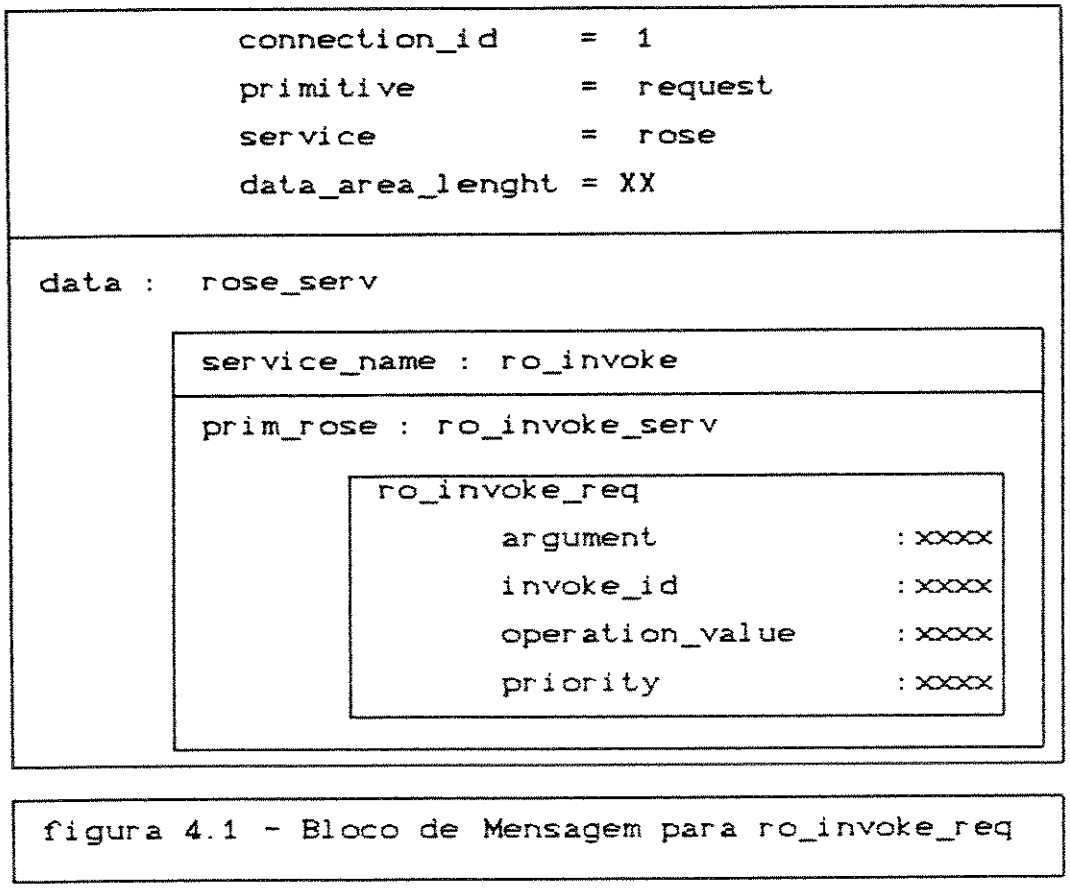


figura 4.1 - Bloco de Mensagem para ro_invoke_req

Após o preparo da estrutura acima, esta deve ser enviada à porta do processo ROSE responsável pelas invocações de operações, através de uma chamada à rotina do núcleo envia_mens. Estas tarefas cabem ao implementador de um SASE que seja usuário do ROSE;

3º) Ao fazer uma chamada à rotina espera_mens, o processo ROSE verifica a existência de mensagens na sua porta de invocações, MSG2_FROM_SASE e, ao encontrar alguma mensagem, executa a ação TREAT_SASE junto com o argumento de chamada P_INVOKES que, por sua vez, faz com que a ação TAKE_MSG2 receba na porta de comunicicação com o SASE, um apontador para o bloco de mensagens do protocolo. Ou seja:

. MSG2_FROM_SASE verdadeira implica em:

```
(!(espera_mens(mb1_sase,WAIT_TIME,VERIFICA,block)
   ==OPERACAO_OK) &&
(espera_mens(mb2_sase,WAIT_TIME,VERIFICA,block)
   ==OPERACAO_OK));
```

. TAKE_MSG2 corresponde a:

```
espera_mens(mb2_sase,WAIT_TIME,CONSOME,block)==OPERACAO_OK;
```

4º) De posse do apontador, a ação DET_SASE_MSG associa um evento à primitiva recebida: evento RO_INV_REQ;

5º) Identificando, no cabeçalho do bloco, a conexão relacionada com a primitiva, a ação TAKE_STATE busca o estado da ROPM na tabela de controle de conexões e o atribui à variável ROPM_STATE. suponhamos que seja STA02;

6º) Com o evento definido e o estado atual da máquina de protocolos, seleciona-se a ação específica que irá tratá-lo de forma a executar o conjunto de ações especificado nos cruzamentos de linhas (evento) e colunas (estado) da tabela de eventos/estados do padrão ISO 9072/2: TREAT_RO_INV_REQ.

7º) A ação TREAT_RO_INV_REQ deverá formar uma APDU ROIV, enviá-la a ROPM_TR e manter o estado STA02. Isto ocorre da seguinte maneira:

. **Formação da APDU ROIV pela ação FORM_ROIV:**

```
block->data.pres_serv.prim_pres.p_data_serv.p_data_req.
        user_data.apdu.roiv.invoke_id=
        block->data.rose_serv.prim_rose.ro_invoke_serv.
        ro_invoke_req.invoke_id;
```

```
if (block->data.rose_serv.prim_rose.ro_invoke_serv.
        ro_invoke_req.linked_id_valid)
        block->data.pres_serv.prim_pres.p_data_serv.p_data_req.
        user_data.apdu.roiv.linked_id=
        block->data.rose_serv.prim_rose.ro_invoke_serv.
        ro_invoke_req.linked_id;

block->data.pres_serv.prim_pres.p_data_serv.p_data_req.
        user_data.apdu.roiv.linked_id_valid=
        block->data.rose_serv.prim_rose.ro_invoke_serv.
        ro_invoke_req.linked_id_valid;

block->data.pres_serv.prim_pres.p_data_serv.p_data_req.
        user_data.apdu.roiv.operation_value=
        block->data.rose_serv.prim_rose.ro_invoke_serv.
        ro_invoke_req.operation_value;

if (block->data.rose_serv.prim_rose.ro_invoke_serv.
        ro_invoke_req.argument_valid)
        block->data.pres_serv.prim_pres.p_data_serv.
        p_data_req.user_data.apdu.roiv.argument=
        block->data.rose_serv.prim_rose.ro_invoke_serv.
        ro_invoke_req.argument;

 block->data.pres_serv.prim_pres.p_data_serv.p_data_req.
        user_data.apdu.roiv.argument_valid=
        block->data.rose_serv.prim_rose.ro_invoke_serv.
        ro_invoke_req.argument_valid;

 block->data.pres_serv.service_name= p_data;

 block->data.pres_serv.prim_pres.p_data_serv.p_data_req.
        user_data.tipo=ROIV;

 block->data.pres_serv.prim_pres.p_data_serv.p_data_req.
        user_data.valid_apdu=TRUE;
```

. **Envio da APDU para ROPM_TR**

Uma primitiva de transferência interna para a ROPM_TR é representada pela ação EXEC_TRANS_REQ. Esta simplesmente envia uma primitiva P_DATA_REQ e define o estado da ROPM_TR como STA210, ou seja, associado e em transferência.

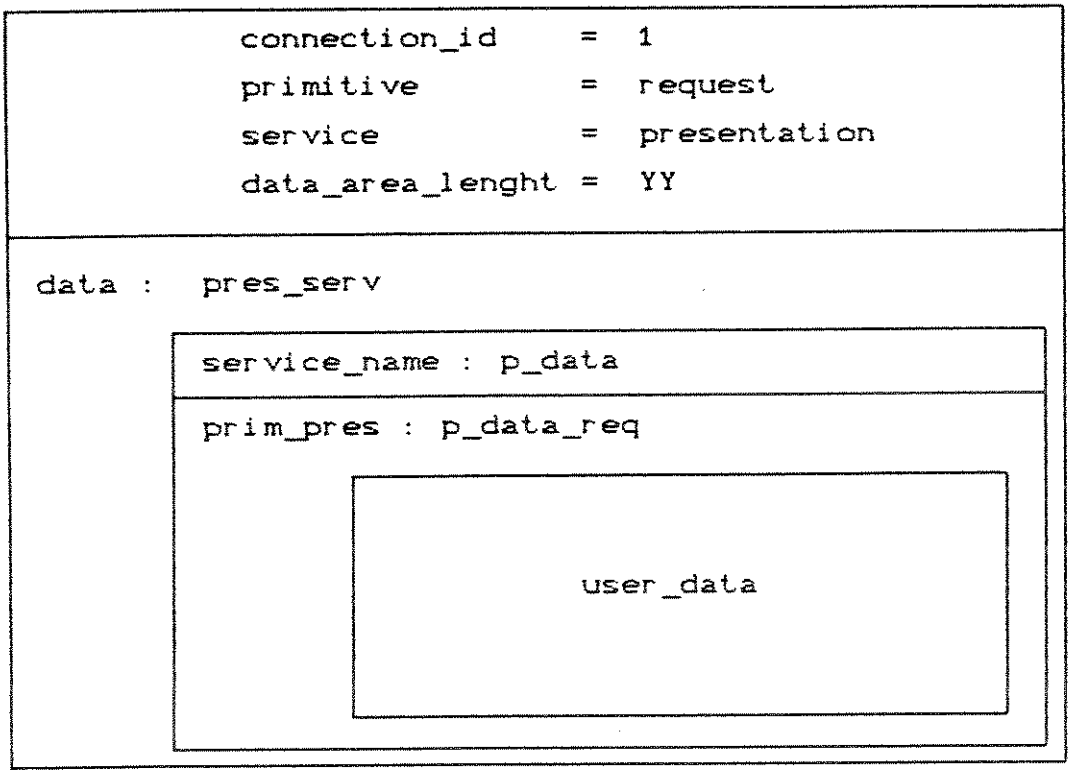


figura 4.2 - Bloco de Mensagem para p_data_req

A APDU ROIV é transportada dentro do parâmetro *user_data* que foi preenchido durante a execução da ação FORM_ROIV.

A operação do processo PROC_ROSE é análoga para as demais APDU's: RORS, ROER e RORJ, mudando apenas alguns parâmetros e portas de entrada.

Tratamento de Mensagem do Provedor

Suponhamos agora que uma mensagem chegue da camada de apresentação. Neste caso, a condição MSG_FROM_PRES será verdadeira e implicará na execução da ação TREAT_PRES. A mensagem suposta é uma primitiva "p_data_ind".

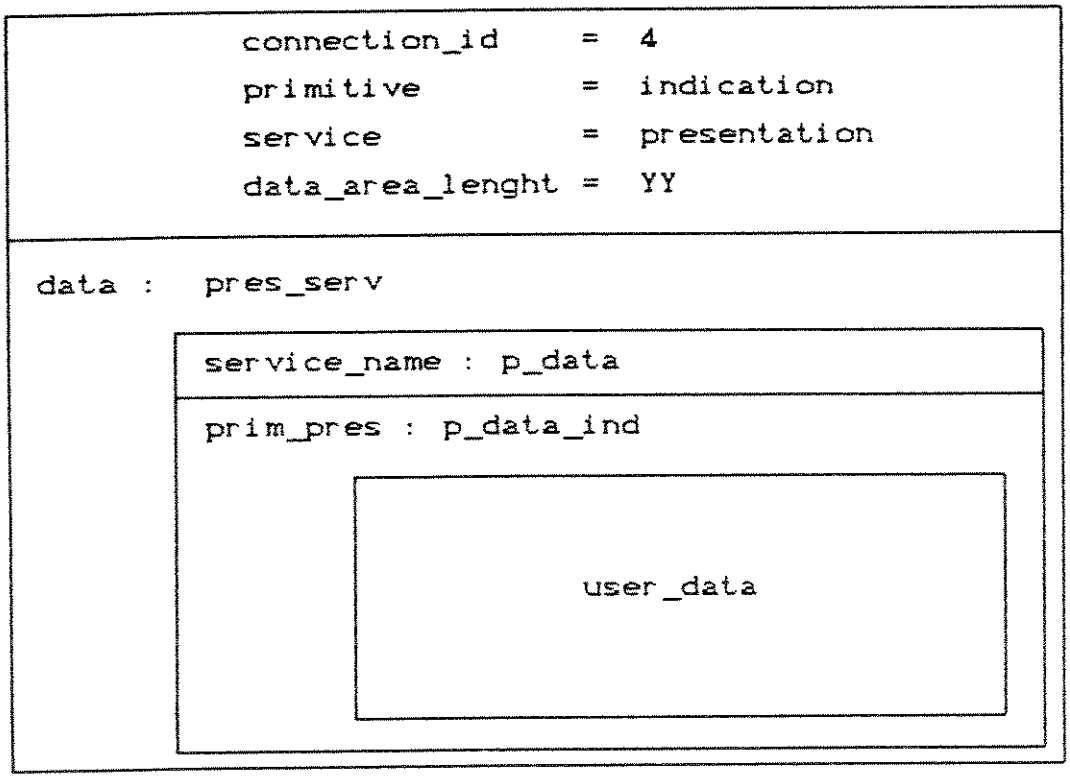


figura 4.3 - Bloco de Mensagem para p_data_ind

Os seguintes passos seriam seguidos para tratar a mensagem da figura 4.3:

1º) Obtenção do apontador para o bloco de mensagens através da chamada:
(espera_mens(mbi_pres,WAIT_TIME,CONSOME,block)==OPERACAO_OK)
que forma a ação TAKE_MSG3.

2º) Identificação do evento e conexão correspondente à mensagem obtida, de forma análoga ao primeiro caso, usando a ação DET_PRES_MSG e o cabeçalho do bloco de mensagens;

2º) Determinação do estado da máquina de protocolos do ROSE (ROPM_TR) através da ação TAKE_STATE. Suponha STA210;

3º) Execução de TREAT_PRES para disparar a ação TREAT_P_DATA_IND que trata o cruzamento evento/estado correspondente a P_DATA_IND e STA210;

4º) Envio de primitiva de transferência interna que comunique à ROPM a chegada de uma APDU da entidade da aplicação remota: EXEC_TRANS_IND.

Em EXEC_TRANS_IND, a APDU contida em *user_data* é verificada de forma que se for inválida, um evento APDUUA é gerado. As possíveis ocorrências com suas respectivas ações de tratamento e indicações para o usuário SASE, são:

a) ROIV	-	TREAT_ROIV	-	SEND_RO_INV_IND;
b) RORS	-	TREAT_RORS	-	SEND_RO_RES_IND;
c) ROER	-	TREAT_ROER	-	SEND_RO_ERR_IND;
d) RORJu	-	TREAT_RORJu	-	SEND_RO_RJU_IND;
e) RORJP	-	TREAT_RORJp	-	SEND_RO_RJP_IND;
f) APDUA	-	TREAT_INVALID_APDU	-	SEND_RO_RJP_IND ou EXEC_AA_AB_REQ;

Supondo que nossa APDU seja válida e sendo a mesma uma invocação de operação, a ação que tratará nossa APDU é a TREAT_ROIV. Considerando que o estado da máquina ROPM seja STA02, será executada a ação SEND_RO_INV_IND que preenche e envia uma primitiva "ro_invoke.ind":

```
  block->primitive=indication;
  block->service=rose;
  block->data.rose_serv.service_name=ro_invoke;

  block->data.rose_serv.prim_rose.ro_invoke_serv.ro_invoke_ind.
         invoke_id=
         block->data.pres_serv.prim_pres.p_data_serv.p_data_ind.
         user_data.apdu.roiv.invoke_id;

  if (block->data.pres_serv.prim_pres.p_data_serv.p_data_ind.
      user_data.apdu.roiv.linked_id_valid)
      block->data.rose_serv.prim_rose.ro_invoke_serv.ro_invoke_ind.
      linked_id= block->data.pres_serv.prim_pres.p_data_serv.
      p_data_ind.user_data.apdu.roiv.linked_id;

block->data.rose_serv.prim_rose.ro_invoke_serv.ro_invoke_ind.
        linked_id_valid=
        block->data.pres_serv.prim_pres.p_data_serv.p_data_ind.
        user_data.apdu.roiv.linked_id_valid;

block->data.rose_serv.prim_rose.ro_invoke_serv.ro_invoke_ind.
        operation_value=
        block->data.pres_serv.prim_pres.p_data_serv.p_data_ind.
        user_data.apdu.roiv.operation_value;

if (block->data.pres_serv.prim_pres.p_data_serv.p_data_ind.
     user_data.apdu.roiv.argument_valid)
    block->data.rose_serv.prim_rose.ro_invoke_serv.ro_invoke_ind.
    argument=  block->data.pres_serv.prim_pres.p_data_serv.
    p_data_ind.user_data.apdu.roiv.argument;

block->data.rose_serv.prim_rose.ro_invoke_serv.ro_invoke_ind.
        argument_valid= block->data.pres_serv.prim_pres.p_data_serv.
        p_data_ind.user_data.apdu.roiv.argument_valid;

  envia_mens(mbo_sase,block);
```

5. CONCLUSÕES

Os padrões ISO 9072 1 e 2, que tratam da especificação do ROSE e seus serviços, provocaram um impacto profundo no modelo OSI. Este fato se deu mais pela notação que os documentos incorporam do que pelo protocolo definido em si. Pois, do que foi exposto, após a definição do ROSE, todos os projetistas de sistemas de comunicação abertos deverão, necessariamente, definir sua aplicação específica através da notação das operações remotas. Isto lhes permitirá uma maior independência da estrutura de primitivas que o modelo tem, como também facilidades com relação à descrição das interações inerentes à aplicação descrita.

O ambiente no qual deverá ser feita a integração do processo ROSE, o SISDI_MAP, é um sistema bastante limitado e que apenas simula a comunicação entre dois sistemas abertos remotos, sendo seu objetivo puramente didático.

A implementação através da ferramenta EPOS considerou todos os serviços apresentados nos padrões disponíveis, mostrando-se bastante útil pela rapidez de desenvolvimento que propiciou por oferecer facilidades tais como:

- desenvolvimento estruturado (TOP-DOWN);
- especificação de implementação especificação de implementação independente de uma linguagem específica;
- análise funcional desta especificação antes mesmo de se determinar uma linguagem de programação final;
- geração de toda documentação, grafica ou textual, de forma dinâmica, ou seja, durante o período de desenvolvimento e
- geração semi-automática de código.

O EPOS e seus módulos componentes, no entanto, ainda têm algumas limitações que diminuem sua performance e aumentam o tempo de projeto como um todo. Entre elas estão:

- demasiadamente lento no ambiente DOS com IBM PC XT/AT ;
- ausência, na especificação S, de atribuições independentes de uma linguagem;
- ausência,na especificação S, de objetos de projeto que representassem funções;
- dependendo da linguagem a ser adotada como linguagem final, apenas alguns objetos de projeto podem ser mapeados diretamente em segmentos de código da mesma;
- o objeto de projeto DATA TYPE é bastante limitado, pois não pode representar em EPOS-S a estrutura de comunicação do SISDI_MAP, bem como alguns outros de seus tipos de dados, tais como *union's* e tipos compostos que os envolvessem.

Com relação ao tempo gasto no desnevolvimento do processo ROSE, pode-se dizer que foi bem menor que o tempo gasto nos primeiros projetos do SISDI_MAP que utilizaram o EPOS, Apresentação e Sessão. A razão para isto é razoavelmente evidente, pois uma vez que os problemas abordados são semelhantes, todos consistiam em implementações de protocolos OSI, alguns artifícios utilizados nos projetos anteriores puderam ser incorporados a este projeto. No entanto, todo protocolo tem suas características especiais que exigem soluções próprias e criação de novos artifícios que poderão ser utilizados pelos trabalhos futuros dentro do sistema didático, diminuindo, assim cada vez mais, o tempo de desenvolvimento de projetos com o EPOS.

Outro aspecto que deve ser comentado é o fato de não existir nos três últimos protocolos implementados as tarefas de codificação e decodificação de PDU's, indispensáveis para que a implementação seja considerada *ABERTA* (desde que as PDU's estejam em ASN.1). No SISDI_MAP isto está previsto através da anexação de um módulo de Codificação/Decodificação de PDU's que executará tais funções a partir de simples chamadas dos protocolos envolvidos. Atualmente a passagem de PDU's no projeto SISDI_MAP não obedece no rigor o especificado no modelo RM_OSI da ISO.

6. Referências Bibliográficas

[1] MAP - "Manufacturing Automation Protocol", versão de especificação 3.0, "Implementation Release - Subject to Errata Changes", abril de 1987.

[2] MENDES, M.J. et al., *SISDI_MAP: SISTEMA DIDÁTICO DO PROTOCOLO E DA INTERFACE DE APLICAÇÃO MMS DO MAP*, Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos, Florianópolis-Brasil, 1989.

[3] ISO 9072/1, *Information processing system - Text communication - Remote Operation - Model, notation and service definition*, 1988.

[4] ISO 9066/1, *Information processing system - Text communication - Remote Operation - Model and Service definition*, 1988.

[5] LAUBER, R.J. e Lemp, P.R., *EPOS - Overview*, 1986.

[6] ISO 8326, *Information processing system - Text communication - Basic Connection Oriented Session Service Definition*, 1988.

[7] ISO 8822, *Information processing system - Text communication - Basic Connection Oriented Presentation Service Definition*, 1988.

[8] ISO 8072, *Information processing system - Text communication - Basic Connection Oriented Transport Service Definition*, 1988.

[9] ISO 9579, *Information processing system - Text communication - Information Retrieval, Transfer and Management for OSI*, agosto/1988.

[10] ISO DIS 9545 - *Information Processing System - Open System Interconection - Application Layer Structure*, XXXX.

[11] ISO 8649, *Information processing system - Text communication - Service Definition for the Association Control Service*

Element, dezembro/1988.

[12] ISO 9804, *Information processing system - Text communication - Definition of Common Application Service Elements - Commitment, Concurrency and Recovery*, Draft International Standard, 1988.

[13] ROSE, M.T., *The Open Book*, Prentice-Hall, 1989.

[14] ISO 8824, *Information Processing System - Open System Interconection - Specification of Abstract Syntax Notation One (ASN.1)*, 1989

[15] ISO 9072/2, *Information processing system - Text communication - Remote Operation - Protocol definition*, 1988.

[16] GIOZZA, W.F. et al., *REDES LOCAIS DE COMPUTADORES - Protocolos de Alto Nível e Avaliação de Desempenho*, EMBRATEL, McGRAW-HILL, 1986.

[17] SILVA, C.R.M., Implementação do Protocolo das Operações Remotas (ROSE) - *Especificação de Requisitos I*, DCA - UNICAMP, Campinas-SP/Brasil, 1991.

[18] SILVA, C.R.M., Implementação do Protocolo das Operações Remotas (ROSE) - *Especificação de Requisitos II*, DCA - UNICAMP, Campinas-SP/Brasil, 1991.

[19] SILVA, C.R.M., Implementação do Protocolo das Operações Remotas (ROSE) - *Especificação de Implementação I*, DCA - UNICAMP, Campinas-SP/Brasil, 1991.

[20] SILVA, C.R.M., Implementação do Protocolo das Operações Remotas (ROSE) - *Especificação de Implementação II*, DCA - UNICAMP, Campinas-SP/Brasil, 1991.

[21] GPP, EPOS-MANUAL, *Code Transformation C - Composer C*, ver. 5.0, 1989.

[22] ZABEU, M.C.A., *Um modelo para configuração de núcleos para tempo real*. Tese de Mestrado, DCA, FEC, FEE, UNICAMP, 1989.

[23] JACINTHO, D.C.A., *Aspectos de Especificação e Implementação da Estrutura de Mensagens de um Sistema Didático de Map.*, Tese de Mestrado, DCA-FEC-FEE UNICAMP, 1989.

[24] HALSALL, F., *Data Communications, Computer Networks and OSI*, Addison-Wesley, 1988.

[25] TANENBAUM, A.S., *Computer Networks*, 2nd edition, Prentice-Hall, 1988.

[26] GIOZZA, W.F. et al., *REDES LOCAIS DE COMPUTADORES - Tecnologia e Aplicações*, EMBRATEL, McGRAW-HILL, 1986.

[27] SVOBODOVA, L., *Implementing OSI Systems*, IEEE Journal on Selected Areas in Communications, vol 7. no.7, september 1989.

[28] KERNIGHAN, R., *The C programming language*, Addison-Wesley, 1984.

[29] WHITE, J.E., *ASN.1 and ROS: the impact of X.400 on OSI*, IEEE journal on selected areas in communications, vol 7, nº 7, setembro 1989.

[30] ISO 8825, *Information Processing System - Open System Interconection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, 1989..

[31] ZABEU, M.C. et al., *Protocolo RS-511 - Um Modelo de Implementação*, IN 7, SBRC, 1989.

[32] ISO N1494, *Application Layer Structure*, 1986.

[33] ISO DIS 7498, *Information Processing System - Open System Interconection - Basic Reference Model*, XXXX.

[34] SCHILDT, H., *C Avançado - guia do usuário*, McGRAW-HILL, 1987.

[35] MENDES, M.J., *A Pesquisa em Sistemas Distribuídos no Brasil*, Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos, Florianópolis - SC, Brasil, 11-14/setembro/1989.

[36] GPP, EPOS-MANUAL, *Communication System EPOS-C*, ver. 5.0, 1989.

[37] GPP, EPOS-MANUAL, *Specification Language EPOS-R*, ver. 5.0, 1989.

[38] GPP, EPOS-MANUAL, *Specification Language EPOS-S*, ver. 5.0, 1989.

[39] GPP, EPOS-MANUAL, *Analysis System EPOS-A*, ver. 5.0, 1989.

[40] GPP, EPOS-MANUAL, *Documentation System EPOS-D*, ver. 5.0, 1989.

[41] BIEWALD, J. et al., *Application of the Specification Techinique EPOS to a process control problem*, proceedings IFAC/IFIP, 1984.

[42] MENDES, M.J., *Comunicação Fabril e o Projeto MAP/TOP*, IV EBAE, janeiro 1989.

[43] ROSE, M.T., *Transition and Coexistence Strategies for TCP/IP to OSI*, IEEE Journal on Selected Areas in Communications, vol 8, nº 1, janeiro 1990.

[44] Svobodova, L. et al. (1990), *Heterogeneity and OSI*, IEEE journal on selected areas in communications, vol.8, nº 1, janeiro, 1990.

Apêndice A
Listagem dos Includes Referentes ao ROSE

```
/*****************************************************************************
                    TIPOS   ESTRUTURADOS  DO  ROSE
 *****************************************************************************/

/*-------------------------------------------------------------
                        Tipos Auxiliares
  -------------------------------------------------------------*/

typedef int  Oper_class_type;

typedef char *Argument_type;

typedef char *Result_type;

typedef char *Error_par_type;


/*
---------------------------------------------------------------
                        ROIV.
---------------------------------------------------------------*/

typedef struct {

/*02*/        uint16              invoke_id;
/*01*/        Boolean             linked_id_valid;
/*02*/        uint16              linked_id;
/*02*/        uint16              operation_value;
/*01*/        Boolean             argument_valid;
/*04*/        Argument_type       argument;

/* 12 */       } Roiv_apdu;

/*
---------------------------------------------------------------
                        RORS.
---------------------------------------------------------------*/

typedef struct {

/*02*/          uint16              operation_value;
/*02*/          uint16              invoke_id;
/*01*/          Boolean             result_valid;
/*04*/          Result_type         result;

/* 09 */       } Rors_apdu;

/*
---------------------------------------------------------------
                        ROER.
---------------------------------------------------------------*/

typedef struct {

/*02*/          uint16              invoke_id;
/*02*/          uint16              error_value;
/*01*/          Boolean             error_parameter_valid;
/*04*/          Error_par_type      error_parameter;

/* 09 */       } Roer_apdu;



/*-------------------------------------------------------------
                        RORJ.
---------------------------------------------------------------*/
```

```
typedef struct {
                        uint16          invoke_id;

                union {
                        General_problem         g_problem;
                        Reject_reason           r_reason;

                      } opcao;

              } Rorj_apdu;

/*-----------------------------------------
                ROSE Apdus
  ---------------------------------------*/
typedef struct {
              Boolean     valid_apdu;
              int         tipo;
              union {
                      Roiv_apdu     roiv;
                      Rors_apdu     rors;
                      Roer_apdu     roer;
                      Rorj_apdu     rorj;
                    }apdu;
             }Rose_apdu_struct;

/*-----------------------------------------
        Estrutura do servico "RO_INVOKE"
  ---------------------------------------*/

/* Obs : Os campos dummy sao reservados para parametros que so'
 *       existem nas primitivas indication/confirm
 */

/*-----------------------------
  Primitiva RO_INVOKE_REQUEST
  -----------------------------*/

typedef struct
        {
/*01*/  Boolean                 operation_class_valid;
/*02*/  Oper_class_type         operation_class;
/*01*/  Boolean                 priority_valid;
/*01*/  uint8                   priority;
/*02*/  uint16                  invoke_id;
/*01*/  Boolean                 linked_id_valid;
/*02*/  uint16                  linked_id;
/*02*/  uint16                  operation_value;
/*01*/  Boolean                 argument_valid;
/*04*/  Argument_type           argument;

        } Ro_inv_req_struct;
                                /* Tamanho dos parametros :  4 bytes
                                 * Tamanho PDU associada  : 12 bytes
                                 * Tamanho Total          : 16 bytes
                                 */

/*-------------------------------
  Primitiva RO_INVOKE_INDICATION
  -------------------------------*/


typedef struct
        {
/*04*/  Dummy                   dummy_ind[04];
/*02*/  uint16                  invoke_id;
/*01*/  Boolean                 linked_id_valid;
/*02*/  uint16                  linked_id;
```

```
/*02*/  uint16                  operation_value;
/*01*/  Boolean                 argument_valid;
/*04*/  Argument_type           argument;

        ) Ro_inv_ind_struct;
                                /* Tamanho dos parametros :  4 bytes
                                 * Tamanho PDU associada  : 12 bytes
                                 * Tamanho Total          : 16 bytes
                                 */


/*-------------------------------------------------------------
      Estrutura do servico de Aplicacao : RO_INVOKE
  -------------------------------------------------------------*/

typedef union
        {
        Ro_inv_req_struct       ro_invoke_req;
        Ro_inv_ind_struct       ro_invoke_ind;
        } Ro_inv_struct;        /* tamanho max : 16 bytes */



/*---------------------------
  Primitiva RO_RESULT_REQUEST
  ---------------------------*/

typedef struct
        {
/*01*/  Boolean                 priority_valid;
/*01*/  uint8                   priority;
/*02*/  uint16                  operation_value;
/*02*/  uint16                  invoke_id;
/*01*/  Boolean                 result_valid;
/*04*/  Result_type             result;
        } Ro_res_req_struct;
                                /* Tamanho dos parametros      :  2 bytes
                                 * Tamanho Max. PDU associada  :  9 bytes
                                 * Tamanho Total               : 11 bytes
                                 */



/*-----------------------------
  Primitiva RO_RESULT_INDICATION
  -----------------------------*/

typedef struct
        {
/*02*/  Dummy                   dummy_ind[02];
/*02*/  uint16                  operation_value;
/*02*/  uint16                  invoke_id;
/*01*/  Boolean                 result_valid;
/*04*/  Result_type             result;

        } Ro_res_ind_struct;
                                /* Tamanho
                                 dos parametros      :  2 bytes
                                 * Tamanho Max. PDU associada  :  9 bytes
                                 * Tamanho Total               : 11 bytes
                                 */



/*-------------------------------------------------------------
      Estrutura do servico de Aplicacao : RO_RESULT
  -------------------------------------------------------------*/

typedef union
```

```
        {
        Ro_res_req_struct       ro_result_req;
        Ro_res_ind_struct       ro_result_ind;
        } Ro_res_struct;        /* tamanho max : 11 bytes */


/*-------------------------
  Primitiva RO_ERROR_REQUEST
  -------------------------*/

typedef struct
        {
/*01*/  Boolean                 priority_valid;
/*01*/  uint8                   priority;
/*02*/  uint16                  invoke_id;
/*02*/  uint16                  error_value;
/*01*/  Boolean                 error_parameter_valid;
/*04*/  Error_par_type          error_parameter;

        } Ro_err_req_struct;
                               /* Tamanho dos parametros      :  2 bytes
                                * Tamanho Max. PDU associada  :  9 bytes
                                * Tamanho Total               : 11 bytes
                                */


/*-------------------------
  Primitiva RO_ERROR_INDICATION
  -------------------------*/

typedef struct
        {
/*02*/  Dummy                   dummy_ind[02];
/*02*/  uint16                  invoke_id;
/*02*/  uint16                  error_value;
/*01*/  Boolean                 error_parameter_valid;
/*04*/  Error_par_type          error_parameter;
        } Ro_err_ind_struct;
                               /* Tamanho dos parametros      :  2 bytes
                                * Tamanho Max. PDU associada  :  9 bytes
                                * Tamanho Total               : 11 bytes
                                */


/*-----------------------------------------------------------
      Estrutura do servico de Aplicacao : RO_ERROR
  -----------------------------------------------------------*/

typedef union
        {
        Ro_err_req_struct       ro_error_req;
        Ro_err_ind_struct       ro_error_ind;
        } Ro_err_struct;        /* tamanho max : 11 bytes */


/*-------------------------
  Primitiva RO_REJECT_U_REQUEST
  -------------------------*/

typedef struct
        {
/*01*/  Boolean                 priority_valid;
/*01*/  uint8                   priority;
/*02*/  Invoke_id               invoke_id;
/*02*/  Reject_reason           r_reason;
        } Ro_rju_req_struct;
                               /* Tamanho dos parametros      :  2 bytes
                                * Tamanho Max. PDU associada  :  4 bytes
                                * Tamanho Total               :  6 bytes
```

```
                                */


/*-------------------------------
  Primitiva RO_REJECT_U_INDICATION
  -------------------------------*/

typedef struct
        {
/*02*/  Dummy                   dummy[2];
/*02*/  Invoke_id               invoke_id;
        Reject_reason           r_reason;
        } Ro_rju_ind_struct;
                               /* Tamanho dos parametros      :  2 bytes
                                * Tamanho Max. PDU associada  :  4 bytes
                                * Tamanho Total               :  6 bytes
                                */


/*-------------------------------------------------------------
      Estrutura do servico de Aplicacao : RO_REJECT_U
  -------------------------------------------------------------*/

typedef union
        {
        Ro_rju_req_struct       ro_rju_req;
        Ro_rju_ind_struct       ro_rju_ind;
        } Ro_rju_struct;        /* tamanho max : 6 bytes */


/*-------------------------------
  Primitiva RO_REJECT_P_INDICATION
  -------------------------------*/

typedef union
        {
/*02*/  Invoke_id               invoke_id;
/*02*/  General_problem         g_problem;
        } Ro_rjp_ind_struct;
                               /* Tamanho dos parametros      :  0 bytes
                                * Tamanho Max. PDU associada  :  4 bytes
                                * Tamanho Total               :  4 bytes
                                */


/*-------------------------------------------------------------
      Estrutura do servico de Aplicacao : RO_REJECT_P
  -------------------------------------------------------------*/

typedef union
        {
        Ro_rjp_ind_struct       ro_rjp_ind;
        } Ro_rjp_struct;        /* tamanho max : 4 bytes */


/*-----------------------------------------------------------------
            Estrutura de Servicos do Protocolo Rose
  -----------------------------------------------------------------*/

typedef struct
        {
         Rose_services                  service_name;
         Dummy                          dummy_rose[396];
         /*
            Dummy                       dummy_rose[DP + DS + DT];
            dummy para os dados das demais camadas inferiores, no caso *
```

```
         * a camada de apresentacao (Dummy DP), a camada de sessao (Dummy DS)
           e transporte (Dummy DT)      */
        union
            {
            Ro_inv_struct           ro_invoke_serv;
            Ro_res_struct           ro_result_serv;
            Ro_err_struct           ro_error_serv;
            Ro_rju_struct           ro_rju_serv;
            Ro_rjp_struct          ro_rjp_serv;
            } prim_rose;
       } Rose_struct;

/*
-------------------------------------------------------------------------
#######       O DUMMY CALCULADO PARA O ROSE E' DE 16 BYTES     #########
-------------------------------------------------------------------------*/
```

Apêndice B
Análises Automáticas Fornecidas pelo EPOS

CHECK ON COMPLETENESS

SPECIFICATION C O M P L E T E !!

****

CHECK ON TYPE-CONFLICTS

SPECIFICATION W I T H O U T TYPE-CONFLICTS !!

****

CHECK ON NOT-REFERENCED DESIGN OBJECTS

N O T E :

SPECIFICATION W I T H NOT-REFERENCED DESIGN OBJECTS !!

NOT-REFERENCED DESIGN OBJECTS:
NAME (TYPE)

CASE_ROSE (A1.1)

CHECK ON UNDEFINED OBJECT PARTS

SPECIFICATION W I T H O U T UNDEFINED OBJECT-PARTS !!

****

CHECK OF ILLEGAL CONSTRUCTIONS IN RELATION TO MODULE SPECIFICATION

SPECIFICATION W I T H O U T ILLEGAL CONSTRUCTIONS !!

CHECK ON MODULE SPECIFICATION

N O T E :

DESIGN OBJECTS ARE IMPORTED BY THE FOLLOWING MODULES, BUT NOT REFERENCED IN THEIR VISIBILITY LAYER. [A-3]

IN MODULE:	DESIGN OBJECT:
ROSE_PROTOCOL (A2.1)	NTR (A3.2)
	STDIO (A3.3)
	SISDI1 (A3.4)
	ROSE_DATA (A3.5)
	RTSE_DATA (A3.6)
	SISDI2 (A3.7)

N O T E :

THE FOLLOWING VISIBILITY LAYERS OF PASSIVE MODULES REFERENCE ACTIVE DESIGN OBJECTS. [A-6]

DESIGN OBJECT:	VISIBILITY LAYER OF THE MODULE:
PROC_ROSE (A3.1)	ROSE_PROTOCOL (A2.1)

MODULE SPECIFICATION IS C O N S I S T E N T !!

****

TEST OF DATA WITHIN REPEAT - CONSTRUCTIONS

SPECIFICATION W I T H O U T INCONSISTENT REPEAT-CONSTRUCTIONS !!

TEST OF DATA WITHIN SWITCH - CONSTRUCTIONS

SPECIFICATION W I T H O U T INCONSISTENT SWITCH-CONSTRUCTIONS !!

TEST OF DATA ACCORDING TO PROCEDURE - CALLS

SPECIFICATION W I T H O U T INCONSISTENT PROCEDURE-CALLS !!

CHECK ON FUNCTIONAL-ISOLATED DESIGN OBJECTS

E R R O R :

A L L DESIGN-OBJECTS ARE FUNCTIONALLY-ISOLATED,
BECAUSE NO EVENT/SYSTEMSTART IS USED !!

CHECK ON HIERARCHICAL COMPLETENESS OF THE DATAFLOW-SPECIFICATION

N O T E :

DATAFLOW HIERARCHICALLY N O T COMPLETE !!

THE FOLLOWING INPUT-/OUTPUT-DATA (OR THEIR COMPLETE DECOMPOSITION) OF AN ACTION DOES NOT EXIST FOR ACTIONS AT THE NEXT LOWER LEVEL.

ACTION	INPUT-DATA
ERROR_MANAGEMENT (A5.5)	ERROR_CODE (D38)
EXEC_TRANS_IND (A6.15)	TRANSFER_MSG2 (D49)
LAST_STATE (A5.8)	NEW_STATE (D40) MACHINE1 (D41)
ROSE_PROTOCOL (A2.1)	MAX_CONTEXT (D1) NONE (D2) SASE_MSG (D3) RTSE_MSG (D4)

TRANSFER_MSG (D5)
ROPM_STATE (D6)
REJECT_NUM (D7)
PRIORITY (D8)
ROPM_TR_STATE (D9)
MAX_REJECT_NUM (D10)
TRANSFER_TIME (D11)
WAIT_TIME (D12)
GLOBAL_VARS (D13)

ACTION	INPUT-DATA
TAKE_STATE (A5.13)	MACHINE4 (D45)

ACTION	INPUT-DATA
TREAT_SASE (A4.2)	SASE_PORT (D16)

****

==========
CHECK ON INTERFACE-INCONSISTENCIES
==========

SPECIFICATION W I T H O U T INTERFACE-INCONSISTENCIES !!

ANALYSIS OF CLOCK AND CYCLE SPECIFICATIONS IN EVENTS

DATA REFERENCE IN EVENTS C O N S I S T E N T !!

UNICAMP

Cidade Universitaria Zeferino Vaz de Almeida

13083 CAMPINAS, SP, BRASIL

PROJECT: CASE_ROSE

USER: C.R.MUNIZ

DATE OF OUTPUT: 3-24-1991

DOCUMENT TYPE: References to formal insertions

Correct references to formal insertions

REQUIREMENT 1(1) fulfilled in

ACTION TREAT_SASE		[13]

REQUIREMENT 2(1) fulfilled in

ACTION ROSE_DATA	partly	[8]
ACTION TREAT_SASE	partly	[13]

REQUIREMENT 3(1) fulfilled in

ACTION TREAT_AA_ESTAB		[14]

REQUIREMENT 4(1) fulfilled in

ACTION SEND_RO_RJP_IND		[28]

REQUIREMENT 5(1) fulfilled in

ACTION SEND_RO_RJU_IND	partly	[53]
ACTION TREAT_RO_RJU_REQ	partly	[25]

REQUIREMENT 6(1) fulfilled in

ACTION SEND_RO_RJP_IND		[28]

REQUIREMENT 7(1) fulfilled in

ACTION PROC_ROSE		[7]

REQUIREMENT 8(1) fulfilled in

ACTION ROSE_DATA		[8]

REQUIREMENT 9(1) fulfilled in

ACTION EXEC_AA_AB_REQ		[54]

REQUIREMENT 10(1) fulfilled in

ACTION TREAT_RO_ERR_REQ	partly	[25]
ACTION TREAT_RO_RES_REQ	partly	[24]

REQUIREMENT 1(2) fulfilled in

ACTION EXEC_TRANS_REQ	partly	[26]
ACTION TREAT_RO_INV_REQ	partly	[23]

REQUIREMENT 2(2) fulfilled in

ACTION SEND_RO_INV_IND	partly	[50]
ACTION TREAT_ROIV	partly	[46]

REQUIREMENT 3(2) fulfilled in

ACTION EXEC_TRANS_REQ	partly	[26]
ACTION TREAT_RO_RES_REQ	partly	[24]

REQUIREMENT 4(2) fulfilled in

ACTION EXEC_TRANS_IND	partly	[38]
ACTION TREAT_INVALID_APDU	partly	[48]
ACTION TREAT_RORS	partly	[46]

REQUIREMENT 5(2) fulfilled in

ACTION EXEC_TRANS_REQ	partly	[26]
ACTION TREAT_RO_ERR_REQ	partly	[25]

REQUIREMENT 6(2) fulfilled in

ACTION EXEC_TRANS_IND	partly	[38]
ACTION SEND_RO_ERR_IND	partly	[52]
ACTION TREAT_ROER	partly	[47]

REQUIREMENT 7(2) fulfilled in

ACTION EXEC_TRANS_REQ	partly	[26]
ACTION TREAT_RO_RJU_REQ	partly	[25]

REQUIREMENT 8(2) fulfilled in

ACTION EXEC_TRANS_IND	partly	[38]
ACTION TREAT_RORJU	partly	[47]

REQUIREMENT 9(2) fulfilled in

ACTION TREAT_RORJP		[48]

REQUIREMENT 10(2) fulfilled in

ACTION TREAT_RORJP		[48]

REQUIREMENT 11(2) fulfilled in

ACTION TREAT_RORJP		[48]

REQUIREMENT 12(2) fulfilled in

ACTION ROSE_DATA		[8]

REQUIREMENT 13(2) fulfilled in

ACTION EXEC_TRANS_IND	partly	[38]
ACTION EXEC_TRANS_REQ	partly	[26]

REQUIREMENT 1(3) [C 3-2] fulfilled in

ACTION ERROR_MANAGEMENT	partly	[22]
ACTION EXEC_AA_AB_IND	partly	[38]
ACTION LAST_STATE	partly	[24]
ACTION TREAT_AA_ESTAB	partly	[14]
ACTION TREAT_AA_REL	partly	[15]
ACTION TREAT_INVALID_APDU	partly	[48]
ACTION TREAT_ROER	partly	[47]
ACTION TREAT_ROIV	partly	[46]
ACTION TREAT_RORJP	partly	[48]
ACTION TREAT_RORJU	partly	[47]
ACTION TREAT_RORS	partly	[46]
ACTION TREAT_RO_ERR_REQ	partly	[25]
ACTION TREAT_RO_INV_REQ	partly	[23]
ACTION TREAT_RO_RES_REQ	partly	[24]
ACTION TREAT_RO_RJU_REQ	partly	[25]

REQUIREMENT 2(3) [C 3-5] fulfilled in

ACTION ERROR_MANAGEMENT	partly	[22]
ACTION TREAT_INVALID_APDU	partly	[48]
ACTION TREAT_ROER	partly	[47]
ACTION TREAT_ROIV	partly	[46]
ACTION TREAT_RORJP	partly	[48]
ACTION TREAT_RORJU	partly	[47]
ACTION TREAT_RORS	partly	[46]
ACTION TREAT_RO_ERR_REQ	partly	[25]
ACTION TREAT_RO_INV_REQ	partly	[23]
ACTION TREAT_RO_RES_REQ	partly	[24]
ACTION TREAT_RO_RJU_REQ	partly	[25]

REQUIREMENT 3(3) [C 3-7] fulfilled in

ACTION ERROR_MANAGEMENT	partly	[22]
ACTION EXEC_AA_AB_REQ	partly	[54]
ACTION SEND_RO_ERR_IND	partly	[52]
ACTION SEND_RO_INV_IND	partly	[50]
ACTION SEND_RO_RES_IND	partly	[51]
ACTION SEND_RO_RJP_IND	partly	[28]
ACTION SEND_RO_RJU_IND	partly	[53]
ACTION TREAT_INVALID_APDU	partly	[48]
ACTION TREAT_ROER	partly	[47]
ACTION TREAT_ROIV	partly	[46]
ACTION TREAT_RORJP	partly	[48]
ACTION TREAT_RORJU	partly	[47]
ACTION TREAT_RORS	partly	[46]

REQUIREMENT 4(3) [C 3-9] fulfilled in

ACTION ERROR_MANAGEMENT	partly	[22]
ACTION EXEC_AA_AB_IND	partly	[38]
ACTION EXEC_AA_AB_REQ	partly	[54]
ACTION EXEC_TRANS_IND	partly	[38]
ACTION EXEC_TRANS_REQ	partly	[26]
ACTION LAST_STATE	partly	[24]
ACTION SEND_ABORT_REQ	partly	[55]
ACTION SEND_P_DATA_REQ	partly	[36]
ACTION SEND_RQ_RJP_IND	partly	[28]
ACTION TREAT_AA_ESTAB	partly	[14]
ACTION TREAT_AA_REL	partly	[15]
ACTION TREAT_ABORT_IND	partly	[29]
ACTION TREAT_P_DATA_IND	partly	[28]

Only considered formal insertions

No elements found !

Ignored formal insertions

No elements found !

Substitutions of formal insertions

No elements found !

Fulfillment extent of substituted formal insertions

No elements found !

Incomplete references to formal insertions

No elements found !

Redundant references to formal insertions

No elements found !

Substitution and reference to formal insertions

No elements found !

Non-referenced formal insertions

No elements found !

Invalid references with date contradiction

No elements found !

Invalic substitutions with level contradiction

No elements found !

Contradictory references to formal insertions

No elements found !

Referenced but not specified formal insertions

No elements found !

UNICAMP

Cidade Universitaria Zeferino Vaz de Almeida

13083 CAMPINAS, SP, BRASIL

PROJECT: CASE_ROSE

USER: C.R.MUNIZ

DATE OF OUTPUT: 3-04-1991

DOCUMENT TYPE: References to formal insertions

Correct references to formal insertions

REQUIREMENT 1(1) [F 3-1] fulfilled in

ACTION TREAT_SASE

REQUIREMENT 2(1) [F 3-1] fulfilled in

ACTION ROSE_DATA	partly
ACTION TREAT_SASE	partly

REQUIREMENT 3(1) [F 3-2] fulfilled in

ACTION TREAT_AA_ESTAB

REQUIREMENT 4(1) [F 3-2] fulfilled in

ACTION SEND_RO_RJP_IND

REQUIREMENT 5(1) [F 3-2] fulfilled in

ACTION SEND_RO_RJU_IND	partly
ACTION TREAT_RO_RJU_REQ	partly

REQUIREMENT 6(1) [F 3-2] fulfilled in

ACTION SEND_RO_RJP_IND

REQUIREMENT 7(1) [F 3-2] fulfilled in

ACTION PROC_ROSE

REQUIREMENT 8(1) [F 3-2] fulfilled in

ACTION ROSE_DATA

REQUIREMENT 9(1) [F 3-4] fulfilled in

ACTION EXEC_AA_AB_REQ

REQUIREMENT 10(1) [F 3-4] fulfilled in

ACTION TREAT_RO_ERR_REQ	partly
ACTION TREAT_RO_RES_REQ	partly

REQUIREMENT 1(2) [F 3-5] fulfilled in

ACTION EXEC_TRANS_REQ	partly
ACTION TREAT_RO_INV_REQ	partly

REQUIREMENT 2(2) [F 3-5] fulfilled in

ACTION SEND_RO_INV_IND	partly
ACTION TREAT_ROIV	partly

REQUIREMENT 3(2) [F 3-6] fulfilled in

ACTION EXEC_TRANS_REQ	partly
ACTION TREAT_RO_RES_REQ	partly

REQUIREMENT 4(2) [F 3-6] fulfilled in

ACTION EXEC_TRANS_IND	partly
ACTION TREAT_INVALID_APDU	partly
ACTION TREAT_RORS	partly

REQUIREMENT 5(2) [F 3-6] fulfilled in

ACTION EXEC_TRANS_REQ	partly
ACTION TREAT_RO_ERR_REQ	partly

REQUIREMENT 6(2) [F 3-7] fulfilled in

ACTION EXEC_TRANS_IND	partly
ACTION SEND_RO_ERR_IND	partly
ACTION TREAT_ROER	partly

REQUIREMENT 7(2) [F 3-7] fulfilled in

ACTION EXEC_TRANS_REQ	partly
ACTION TREAT_RO_RJU_REQ	partly

REQUIREMENT 8(2) [F 3-7] fulfilled in

ACTION EXEC_TRANS_IND	partly
ACTION TREAT_RORJU	partly

REQUIREMENT 9(2) [F 3-8] fulfilled in

ACTION TREAT_RORJP

REQUIREMENT 10(2) [F 3-8] fulfilled in

ACTION TREAT_RORJP

REQUIREMENT 11(2) [F 3-9] fulfilled in

ACTION TREAT_RORJP

REQUIREMENT 12(2) [F 3-9] fulfilled in

ACTION ROSE_DATA

REQUIREMENT 13(2) [F 3-10] fulfilled in

ACTION EXEC_TRANS_IND	partly
ACTION EXEC_TRANS_REQ	partly

REQUIREMENT 1(3) [C 3-2] fulfilled in

ACTION ERROR_MANAGEMENT	partly
ACTION EXEC_AA_AB_IND	partly
ACTION LAST_STATE	partly
ACTION TREAT_AA_ESTAB	partly
ACTION TREAT_AA_REL	partly
ACTION TREAT_INVALID_APDU	partly
ACTION TREAT_ROER	partly
ACTION TREAT_ROIV	partly
ACTION TREAT_RORJP	partly
ACTION TREAT_RORJU	partly
ACTION TREAT_RORS	partly
ACTION TREAT_RO_ERR_REQ	partly
ACTION TREAT_RO_INV_REQ	partly
ACTION TREAT_RO_RES_REQ	partly
ACTION TREAT_RO_RJU_REQ	partly

REQUIREMENT 2(3) [C 3-5] fulfilled in

ACTION ERROR_MANAGEMENT	partly
ACTION TREAT_INVALID_APDU	partly
ACTION TREAT_ROER	partly
ACTION TREAT_ROIV	partly
ACTION TREAT_RORJP	partly
ACTION TREAT_RORJU	partly
ACTION TREAT_RORS	partly
ACTION TREAT_RO_ERR_REQ	partly
ACTION TREAT_RO_INV_REQ	partly
ACTION TREAT_RO_RES_REQ	partly
ACTION TREAT_RO_RJU_REQ	partly

REQUIREMENT 3(3) [C 3-7] fulfilled in

ACTION ERROR_MANAGEMENT	partly
ACTION EXEC_AA_AB_REQ	partly
ACTION SEND_RO_ERR_IND	partly
ACTION SEND_RO_INV_IND	partly
ACTION SEND_RO_RES_IND	partly
ACTION SEND_RO_RJP_IND	partly
ACTION SEND_RO_RJU_IND	partly
ACTION TREAT_INVALID_APDU	partly
ACTION TREAT_ROER	partly
ACTION TREAT_ROIV	partly
ACTION TREAT_RORJP	partly
ACTION TREAT_RORJU	partly
ACTION TREAT_RORS	partly

REQUIREMENT 4(3) [C 3-9] fulfilled in

ACTION ERROR_MANAGEMENT	partly
ACTION EXEC_TRANS_IND	partly
ACTION EXEC_TRANS_REQ	partly
ACTION LAST_STATE	partly
ACTION SEND_RO_RJP_IND	partly
ACTION SEND_RT_TG_REQ	partly
ACTION SEND_RT_TP_REQ	partly
ACTION SEND_RT_TR_REQ	partly
ACTION TREAT_AA_ESTAB	partly
ACTION TREAT_RT_TP_IND	partly
ACTION TREAT_RT_TR_CNF_NEG	partly
ACTION TREAT_RT_TR_CNF_POS	partly
ACTION TREAT_RT_TR_IND	partly

REQUIREMENT 5(3) [C 3-11] fulfilled in

ACTION ERROR_MANAGEMENT	partly
ACTION EXEC_AA_AB_IND	partly
ACTION EXEC_AA_AB_REQ	partly
ACTION LAST_STATE	partly
ACTION SEND_ABORT_REQ	partly
ACTION SEND_RO_RJP_IND	partly
ACTION TREAT_AA_REL	partly
ACTION TREAT_ABORT_IND	partly

Only considered formal insertions

No elements found !

Ignored formal insertions

No elements found !

Substitutions of formal insertions

No elements found !

Fulfillment extent of substituted formal insertions

No elements found !

Incomplete references to formal insertions

No elements found !

Redundant references to formal insertions

No elements found !

Substitution and reference to formal insertions

No elements found !

Non-referenced formal insertions

No elements found !

Invalid references with date contradiction

No elements found !

Invalid substitutions with level contradiction

No elements found !

Contradictory references to formal insertions

No elements found !

Referenced but not specified formal insertions

No elements found !