DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E CIÊNCIA DA COMPUTAÇÃO
UNICAMP - UNIVERSIDADE ESTADUAL DE CAMPINAS

Um Refinamento da Estrutura da Camada de Aplicação do RM-OSI/ISO e Aspectos de sua Implementação em um Sistema Didático de Comunicação

por Flávio Morais de Assis Silva
orientador : Prof. Dr. Edmundo Roberto Mauro Madeira

Campinas, 27 de maio de 1993

Um Refinamento da Estrutura da Camada de Aplicação do RM-OSI/ISO e Aspectos de sua Implementação em um Sistema Didático de Comunicação

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Flávio Morais de Assis Silva e aprovada pela Comissão Julgadora.

Campinas, 03 de maio de 1993

Prof. Dr. Edmundo R M Madeira

Edmundo Roberto Mauro Madeira

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para obtenção do Título de MESTRE em Ciência da Computação

Dedico este trabalho a

meu pai, José,
minha mãe, Mirza,
e a meus irmãos, Cláudio,
Maria do Carmo e
Fabiano

AGRADECIMENTOS

Por terem contribuído direta ou indiretamente para a realização deste trabalho, agradeço ao Prof. Edmundo Roberto Mauro Madeira, pela sua dedicada orientação e completa disponibilidade e atenção durante todo o perído de desenvolvimento da tese; ao Prof. Manuel de Jesus Mendes, por suas contribuições e total solicitude em colaborar no que fosse preciso; ao Prof. Nelson Castro Machado, por ter fornecido condições para o início deste trabalho; ao Prof. José Marcos Silva Nogueira, da UFMG (Universidade Federal de Minas Gerais), e à FEE (Faculdade de Engenharia Elétrica da Unicamp) pelo fornecimento de ferramentas que puderam ser analisadas e usadas nas implementações; ao grande amigo Célio Toshihiro Fujito, com quem mais diretamente convivi e com quem discuti vários aspectos da tese; ao pessoal do grupo de redes, pelas discussões e troca de idéias nas reuniões e fora delas; aos colegas, professores e funcionários do departamento de Ciência da Computação, pelo agradável convívio durante estes anos; ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), pelo suporte financeiro; e à minha família, a quem devo este trabalho.

RESUMO

Esta dissertação consiste de uma definição mais precisa de funcionalidades para os componentes da camada de Aplicação do RM-OSI/ISO (*Reference Model - Open Systems Interconnection / International Organization for Standardization*). Esta definição de funcionalidades tem a finalidade de esclarecer os relacionamentos que há entre os componentes desta camada e de servir de uma base conceitual, a partir da qual estruturas para implementações possam ser derivadas. As funcionalidades apresentadas constituem um refinamento da estrutura definida pela ISO para esta camada.

Também é apresentada a estrutura geral de implementação dos protocolos para um sistema didático de comunicação, chamado SISDI-OSI (*Sistema Didático para o Modelo OSI*), que segue as padronizações do RM-OSI/ISO. Para este sistema descreve-se, em particular, como as estruturas de dados geradas por um compilador para a linguagem ASN.1 são usadas na implementação destes protocolos.

A dissertação consiste ainda da descrição de uma implementação do protocolo CCR (*Commitment, Concurrency and Recovery*), da camada de Aplicação do RM-OSI/ISO. Esta implementação foi feita utilizando-se a linguagem ESTELLE, que é uma Técnica de Descrição Formal definida pela ISO, e faz parte do SISDI-OSI. A estrutura desta implementação foi feita de acordo com a definição de funcionalidades proposta.

ABSTRACT

This thesis consists of a more accurate definition of functionalities for the components of the Application Layer of the RM-OSI/ISO (*Reference Model - Open Systems Interconnection / International Organization for Standardization*). This definition of functionalities intends to clarify the relationships that exist among the components of this layer and to serve as a basis of concepts, from which structures for implementations could be derived. The presented functionalities are a refinement of the structure defined by ISO for this layer.

It also presents the general structure for the implementation of the protocols of a ditactic communication system, called SISDI-OSI (*Didactic System for OSI Model*), which conforms to the standards from RM-OSI/ISO. For this system it is described, particularly, how the data structures generated by a compiler for the ASN.1 language are used in the implementation of these protocols.

This thesis also describes an implementation of the CCR (*Commitment, Concurrency and Recovery*) protocol, from the Application Layer of RM-OSI/ISO. This implementation was done using the language ESTELLE, that is a Formal Description Technique defined by ISO, and takes part in SISDI-OSI. The structure of this implementation was done according to the proposed functionalities.

Conteúdo

Siglas Usadas vii

1 Introdução 1
1.1 O Modelo de Referência RM-OSI 1
1.2 O SISDI-MAP 3
1.3 O SISDI-OSI 4
1.4 Escopo desta Tese 5

2 Uma Definição de Funcionalidades para a Camada de Aplicação do Modelo RM-OSI 7
2.1 Definições Conceituais da Camada de Aplicação 8
2.1.1 Entidade de Aplicação, Elemento de Serviço de Aplicação e Associação de Aplicação 8
2.1.2 Contexto de Aplicação 10
2.1.3 SAO, SACF e MACF 11
2.2 Interpretação dos Relacionamentos entre os Conceitos 12
2.3 Características de Procedimentos Comuns aos Contextos de Aplicação 15
2.3.1 Endereçamento na Camada de Aplicação 15
2.3.2 Uso dos Serviços de Diretório (DS) para o Suporte a Endereçamento 18
2.3.3 Estabelecimento de Associação 20
2.3.4 Uso dos Recursos das Camadas Inferiores 22
2.4 Estrutura dos Componentes da Camada de Aplicação 24
2.4.1 As AEs 24
2.4.2 AEIs 25
2.4.3 MACF 26
2.4.4 O SACF 28

3 Estrutura Geral de Implementação dos Protocolos do SISDI-OSI 31
3.1 A Estrutura Geral dos Módulos 31
3.2 A Estrutura dos Protocolos em Processos 32
3.3 A Estrutura de Comunicação 34
3.4 A Estrutura da Área Comum (*BUFFER*) 35
3.5 Utilização dos Mecanismos de Comunicação pelos Processos 36
3.5.1 Estrutura das Mensagens na Fila 37

3.5.2 A Estrutura das Áreas Alocadas do *Buffer* : Armazenamento de PDUs e Primitivas 38

4 O ASE CCR (*Commitment, Concurrency and Recovery*) 55
4.1 O Modelo Geral do CCR 56
4.2 Os Serviços e suas Primitivas 60
4.3 Recuperação 65
4.4 Comportamento de um Nó Intermediário 70
4.5 Decisões Heurísticas 71
4.6 O Protocolo CCR 71
4.6.1 Procedimentos do protocolo 71
4.6.2 Colisões 78
4.7 Uso do CCR por um Serviço Principal Cooperativo 80

5 A Implementação do CCR e de Componentes Auxiliares 81
5.1 Os Componentes Implementados e as Interações entre eles 81
5.2 O Uso de ESTELLE e o Ambiente EWS 86
5.3 A Implementação do CCR 88
5.3.1 A Implementação do Módulo *CCR* 89
5.3.2 O Módulo *INTERFACE* 104
5.3.3 O Módulo *ESPECIFICAÇÃO* 106
5.4 A Implementação do SACF 107
5.5 Implementação da Simulação das Camadas Inferiores 116
5.6 Testes das Implementações 119

6 Conclusão 121

A Características de ESTELLE usadas nas Implementações 125
A.1 O Modelo Geral de ESTELLE 126
A.2 Alguns Aspectos de Sintaxe da Linguagem 129
A.2.1 Definição de Canais e Declaração de Pontos de Interação 129
A.2.2 Cabeçalho e Corpo de Módulos 130
A.2.3 Declarações 131
A.2.4 Transições 132
A.2.5 Inicialização 137
A.2.6 O Módulo de Especificação 137

Lista de Figuras

1.1 Estrutura do SISDI-MAP 4
1.2 Estrutura do SISDI-OSI 5

2.1 Relacionamento de comunicação entre entidades no RM-OSI 9
2.2 Sistemas, APIs, AEIs, Associação e Camada de Aplicação 10
2.3 Invocação de AE com MACF, SAOs, SACFs, ASEs e associações 12
2.4 Relacionamento entre APIs, AEIs e SAOs 13
2.5 *Pool* de associações livres 14
2.6 Relação entre SAPs e Camadas 16
2.7 Relacionamento entre AE, PSAPs e Endereço de Apresentação 16
2.8 Usos do protocolo DS para obtenção de informações de endereçamento 19
2.9 Invocação de Serviço de um AP por uma AE 20
2.10 Modelo para AEs 25
2.11 Modelo para AEIs 26
2.12 Modelo para MACFs 27
2.13 Modelo para SAOs 28
2.14 Modelo para a parte específica de contexto de aplicação de SACFs 29

3.1 Esquema Geral de Implementação dos Protocolos 32
3.2 Estrutura geral de mensagens do mecanismo *Messages* 37
3.3 Formato das mensagens colocadas nas filas 37
3.4 Exemplo do uso dos campos das mensagens colocadas nas filas 39
3.5 Mapeamento do tipo *CHOICE* em C 41
3.6 Mapeamento do tipo *SEQUENCE OF EXTERNAL* em C 44
3.7 Definição do tipo *EXTERNAL* pelo compilador ASN.1 44
3.8 Definição do tipo *ANY* pelo compilador ASN.1 45
3.9 Mapeamento do tipo das APDUs do CCR em C 46
3.10 Exemplo do uso dos campos do tipo *ANY* 47
3.11 Exemplo de armazenamento de uma APDU no *buffer* para ser enviada como dado de usuário de uma primitiva de um ASE 48
3.12 Definição do tipo *User_data* em C 50
3.13 Exemplo de armazenamento de uma APDU no *buffer* para ser enviada ao SACF . . 51
3.14 Exemplo de armazenamento de uma APDU no *buffer* para ser enviada como dado de usuário de uma primitiva da camada de Apresentação 52
3.15 Exemplo do uso de uma primitiva de concatenação 53

4.1 Árvore de ação atômica 57
4.2 O protocolo de duas fases 59
4.3 Diagrama de tempo para um caso em que ocorre efetivação 64
4.4 Diagrama de tempo para um caso em que ocorre *rollback* 66
4.5 Casos de recuperação 69
4.6 Caso de colisão de C-ROLLBACK-RI, com reenvio de C-BEGIN-RI 79

5.1 Componentes implementados e seus relacionamentos 82
5.2 Mapeamento de conexões de apresentação 84
5.3 Estrutura de implementação do CCR em ESTELLE 88
5.4 Cabeçalho do módulo *CCR* 90
5.5 Canal para troca de primitivas entre o CCR e seu usuário 91
5.6 Canais para comunicação entre o CCR e o SACF 92
5.7 Máquina de estados implementada 94
5.8 Estados e conjuntos de estados 98
5.9 Inicialização do módulo *CCR* 99
5.10 Transições de controle do módulo *CCR* 100
5.11 Transições para C-ROLLBACK.req 101
5.12 Lista de APDUs 102
5.13 Transição para detecção de erro 103
5.14 Cabeçalho do módulo *INTERFACE* 104
5.15 Transição de conversão da interação C_BEGIN_ind 106
5.16 Transições do módulo *ESPECIFICAÇÃO* 107
5.17 Exemplo de problema causado por falta de sincronismo entre componentes 110
5.18 Estrutura de implementação do SACF 110
5.19 Cabeçalho do módulo *SACF* 111
5.20 Canais usados pelo SACF para comunicação com o *ESTABELECEDOR DE ASSOCIAÇÃO*, com seu usuário e com o CCR 111
5.21 Canal usado pelo SACF para comunicação com a camada de Apresentação 112
5.22 Transição para tratamento das informações iniciais 114
5.23 Transição para tratamento da solicitação de término do SAO 114
5.24 Exemplo de controle sobre recursos das camadas inferiores 115
5.25 Transição de tratamento de P-TYPED-DATA.ind 116
5.26 Transições de tratamento de indicações de erro de protocolo feitas pelo CCR 117
5.27 Estruturas de dados mantidas pelo simulador das camadas inferiores 118
5.28 Tratamento de primitivas com mesmos parâmetros 119
5.29 Tratamento de primitivas com parâmetros diferentes 120

A.1 Hierarquia de módulos e instâncias de módulos ESTELLE 127
A.2 Efeito das classes dos módulos na seleção de transições 128
A.3 Declaração de canais 129
A.4 Cabeçalho de módulo 130
A.5 Corpo de módulo 131
A.6 Declaração de variáveis de módulo 132
A.7 Estados e conjuntos de estados 132

A.8 Transições 133
A.9 Cláusula EXIST 134
A.10 Expansão de uma cláusula ANY 134
A.11 Transições aninhadas 135
A.12 Estrutura de uma especificação em ESTELLE 138

Lista de Tabelas

3.1 Rotinas para manipulação do *Buffer* 36
3.2 Rotinas para manipulação do semáforo para controle de concorrência ao *buffer* . . . 37
3.3 Rotinas para manipulação de filas 39

4.1 Correspondência entre APDUs e primitivas 72
4.2 APDUs do CCR e seus campos 73
4.3 Mapeamento de APDUs do CCR e primitivas 75

5.1 Tabela de estados para a função de superior durante transmissão normal de dados . 97

Siglas usadas

Os números indicam as páginas em que o termo a que a sigla se refere está definido:

ACSE Association Control Service Element – Elemento de Serviço de Controle de Associação, 3

AE Application Entity – Entidade de Aplicação, 8

AEI Application Entity Invocation – Invocação de Entidade de Aplicação, 9

AP Application Process – Processo de Aplicação, 8

APDU Application Protocol Data Unit – Unidade de Dados de Protocolo da Aplicação, 9

API Application Process Invocation – Invocação de Processo de Aplicação, 9

API Application Program Interface – Interface de Programa de Aplicação, 3. No texto será explicitamente indicado quando o uso da sigla API tiver este significado, o que ocorre poucas vezes. Na maior parte do texto esta sigla terá o significado do item anterior

ASE Application Service Element – Elemento de Serviço de Aplicação, 9

ASN.1 Abstract Syntax Notation One – Notação de Sintaxe Abstrata Um, 40

BER Basic Encoding Rules – Regras de Codificação Básicas, 40

CCITT The International Telegraph and Telephone Consultative Committee

CCR Commitment, Concurrency and Recovery – Efetivação, Concorrência e Recuperação, 55

DS Directory Services – Serviços de Diretório, 18

FTAM File Transfer, Access and Management – Transferência, Acesso e Gerenciamento de Arquivo, 9

ISO International Organization for Standardization

MACF Multiple Association Control Function – Função de Controle de Múltiplas Associações, 12

MMS Manufacturing Message Specification – Especificação de Mensagens da Manufatura, 3

NM Network Management – Gerenciamento de Redes, 4

PDU Protocol Data Unit – Unidade de Dados de Protocolo, 8

P-SAP Presentation - SAP – SAP da camada de Apresentação, 15

RDA Remote Database Access – Acesso a Bases de Dados Remotas, 2

RM-OSI Reference Model - Open Systems Interconnection – Modelo de Referência - Interconexão de Sistemas Abertos, 1

ROSE Remote Operations Service Element – Elemento de Serviço para Operações Remotas, 3

SACF Single Association Control Function – Função de Controle de uma Única Associação, 12

SAO Single Association Object – Objeto de uma Única Associação, 11

SAP Service Access Point – Ponto de Acesso a Serviços, 15

SISDI-MAP Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP, 3

SISDI-OSI Sistema Didático para o Modelo OSI, 4

TP Transaction Processing – Processamento de Transações, 56

Capítulo 1

Introdução

1.1 O Modelo de Referência RM-OSI

Grande parte dos sistemas de computação atualmente existentes difere em vários aspectos de suas atividades de processamento, armazenamento e comunicação de informação (representação de dados, sistema de arquivos, etc.). Dois sistemas que diferem em suas estruturas ou convenções são ditos *heterogêneos*.

Inúmeros problemas surgem quando há a necessidade de comunicação entre dois sistemas heterogêneos. Para que a comunicação se processe sem problemas, é preciso que os dois sistemas comunicantes "entendam" não só a estrutura dos dados que forem transmitidos como também a sua semântica. Se, por exemplo, cada sistema possui uma convenção de representação de dados diferente (um representa inteiros em complemento de um e o outro em complemento de dois, um deles utiliza caracteres ASCII e o outro EBCDIC, etc.) mesmo que a transmissão de um dado ocorra sem erros, o sistema receptor não conseguirá "entendê-lo".

A ISO (*International Organization for Standardization*) definiu um modelo de arquitetura, chamado de *Reference Model - Open Systems Interconnection* (Modelo de Referência - Interconexão de Sistemas Abertos), ou, abreviadamente, *RM-OSI*, que tem a finalidade de fornecer um modelo de sistema de comunicação que sirva como base para o desenvolvimento de soluções padrões para os problemas de comunicação entre sistemas heterogêneos ([ISO7498], [Tan88], [Bla89] e [Hal88], [Hen90]). Uma vez estabelecidas estas soluções padrões, cada sistema que as adotar deve seguir regras bem definidas que regulem a comunicação e converter suas representações locais para representações comuns para transmissão (e vice-versa) e será, assim, capaz de se comunicar com qualquer outro sistema que também as adotar, por mais heterogêneo que seja.

O modelo RM-OSI fornece uma estrutura conceitual e funcional que permite que grupos internacionais de especialistas trabalhem separada e independentemente no desenvolvimento de tais padrões. Estes padrões procuram atender às necessidades de comunicação que surgem em aplicações distribuídas, nas quais processos são executados em mais de um sistema de computação, mas desempenham funções que, conjuntamente, formam uma tarefa global.

Este modelo é estruturado em camadas, com a finalidade de dividir o problema de comunicação em funções logicamente similares e compreensíveis, sem que esta divisão introduza complexidade desnecessária [ISO7498]. Cada uma das camadas desempenha funções de comunicação específicas. O nome delas e um resumo de suas funções são:

camada 1: FÍSICA fornece as características mecânicas, elétricas, funcionais e procedurais para se acessar o meio físico de transmissão, para o envio e recebimento efetivo de *bits*;

camada 2: ENLACE trata da transferência de dados entre dois sistemas adjacentes na rede (comunicação *nó-a-nó*), fazendo com que a comunicação se passe como se fosse através de uma linha livre de erros;

camada 3: REDE trata do encaminhamento (*roteamento*) dos dados do nó fonte ao nó destino de comunciação, levando em consideração a topologia da rede. Esta camada permite que as camadas superiores cuidem apenas da transmissão *fim-a-fim*, ou seja, as entidades comunicantes podem comportar-se como se estivessem comunicando diretamente, sem que as mensagens passassem por nós intermediários;

camada 4: TRANSPORTE esta camada vai tratar do fornecimento para as camadas superiores de um serviço de transmissão de dados sem erros e de forma ordenada, procurando atender a parâmetros de qualidade de serviço (probabilidade de falha na transferência de dados, prioridade, tempo máximo de estabelecimento de uma conexão, dentre outros) solicitados pelo usuário. É a camada de Transporte que vai tornar transparente a existência do sistema de transmissão subjacente, sua estrutura e características tecnológicas;

camada 5: SESSÃO esta camada trata da coordenação e sincronização do diálogo entre entidades comunicantes. Controla o sentido permitido de comunicação (se em apenas uma direção, em ambas as direções mas de modo alternado, ou se nos dois sentidos simultaneamente), permite a definição de pontos de sincronização no diálogo, para o restabelecimento da comunicação em caso de falhas, e também a divisão da transmissão em partes logicamente distintas;

camada 6: APRESENTAÇÃO fornece mecanismos para que os dados recebidos em um sistema sejam perfeitamente interpretados, independentemente da representação local que possuam nos sistemas comunicantes. Para isto permite que estes sistemas combinem *sintaxes de transferência*, que definem a forma comum de representação dos dados para a transmissão. Ambos os sistemas comunicantes devem suportar as sintaxes de transferência combinadas, de forma que cada um deles "compreenda" os dados que forem transmitidos. Funções de compressão e encriptação de dados também podem ser realizadas nesta camada;

camada 7: APLICAÇÃO esta é a camada mais alta do modelo, fornecendo todos os serviços do sistema de comunicação ao usuário. Os serviços desta camada são mais próximos das necessidades de comunicação que surgem em suas aplicações. Podem ser citados como exemplos de tais serviços o suporte a operações sobre arquivos ou bases de dados remotos e serviço de correio eletrônico.

Cada uma destas camadas possui um conjunto de *protocolos*, que definem regras e convenções para a troca de dados entre os sistemas para um determinado fim específico, e são os mecanismos através dos quais as funções da camada são desempenhadas. Dentre as camadas, a de Aplicação se destaca por possuir um grande conjunto de protocolos, com padrões de relacionamento complexos. Dentre estes protocolos podem-se citar como exemplos o RDA (*Remote Database Access*), para acesso a bancos de dados remotos, o FTAM (*File Transfer, Access and Management*), para acesso

a arquivos em outros sistemas, e o TP (*Transaction Processing*) e CCR (*Commitment, Concurrency and Recovery*), para a execução de transações atômicas distribuídas.

1.2 O SISDI-MAP

Para resolver os problemas de comunicação específicos de ambientes da manufatura, algumas empresas interessadas se uniram e definiram o projeto MAP (*Manufacturing Automation Protocol*) [Men90]. Embora com características próprias, o modelo deste projeto é fortemente baseado no RM-OSI, utilizando seus conceitos e sua estrutura em camadas. Com o objetivo de obter experiência na implementação de protocolos e de possuir um sistema de comunicação que servisse como uma ferramenta didática a ser usada em aulas práticas dos cursos de redes de computadores, o Departamento de Engenharia da Computação e Automação Industrial (DCA) da Faculdade de Engenharia Elétrica da Unicamp criou o projeto SISDI-MAP (*Sistema Didático do Projeto MAP*) [Pag89]. A estrutura inicial do SISDI-MAP aparece na figura 1.1, e consistia:

- do protocolo ACSE (*Association Control Service Element*), que é o protocolo da camada de Aplicação responsável pelo estabelecimento e liberação de *associações*. Para os protocolos até agora definidos para esta camada, uma associação deve sempre ser estabelecida entre os sistemas comunicantes antes que se envie qualquer mensagem;
- do protocolo MMS (*Manufacturing Message Specification*), também um protocolo da Aplicação, que fornece mecanismos para a transferência de mensagens no ambiente da manufatura;
- da API (*Application Program Interface*) do protocolo MMS, definida no projeto MAP, que serve de interface entre este protocolo e seu usuário (um *Processo de Aplicação*);
- Processos de Aplicação (indicados por *APs*, de *Application Processes*), que representam as aplicações dos usuários;
- uma *INTERFACE DE OPERAÇÃO* com o usuário, que fornece um ambiente para a execução dos APs, permite o acompanhamento das atividades executadas no sistema e a introdução de erros de comunicação, para se analisar situações de exceção. Através desta interface o usuário pode também invocar os serviços de rede diretamente à API, de modo individual, para analisar os seus efeitos no sistema; e
- um *SIMULADOR DOS NÍVEIS INFERIORES*, que executa as funções de transferência dos dados dos protocolos de Aplicação, simulando as camadas inferiores e a rede de comunicação. No SISDI-MAP uma só máquina é usada e simula a comunicação entre várias outras.

Este sistema foi projetado para computadores compatíveis com o IBM-PC e utilizava um núcleo de tempo real, desenvolvido no próprio departamento, para fornecer um ambiente multi-tarefa.

Em uma segunda etapa foram implementados para este sistema os serviços básicos dos protocolos orientados a conexão das camadas de Sessão [Mdz90] e Apresentação [Ina90]. Estes serviços permitem o estabelecimento e a liberação de conexões e a transmissão de dados a nível destas camadas. Na camada de Aplicação foi acrescentado o protocolo ROSE (*Remote Operations Service Element*), para a execução de operações remotas [Mun91].

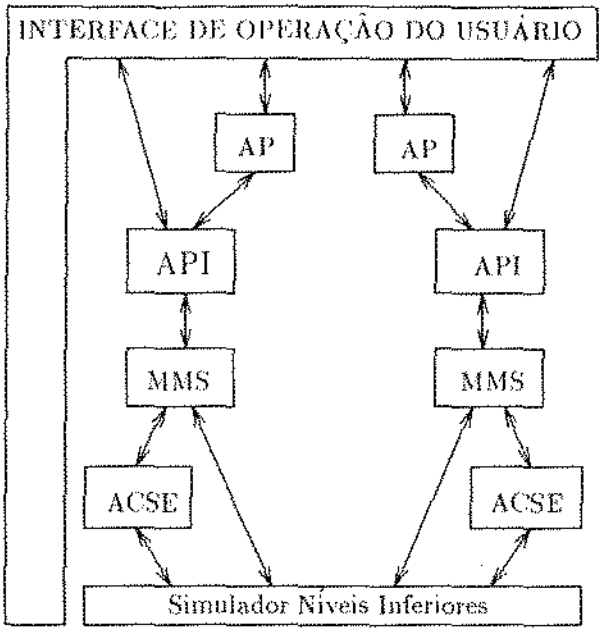

Figura 1.1: Estrutura do SISDI-MAP

1.3 O SISDI-OSI

A nova versão deste sistema, que está sendo agora desenvolvida [Men93], possui grandes diferenças em relação à versão anterior. Embora mantendo os mesmos objetivos básicos, o sistema passou a se basear não mais no projeto MAP, mas no RM-OSI em si (embora a API do protocolo MMS permaneça e só pertença ao projeto MAP). O sistema teve seu nome mudado para SISDI-OSI (*Sistema Didático para o Modelo OSI*) e passou a contar com a participação do Departamento de Ciência da Computação da Unicamp. Esta versão está sendo projetada para o ambiente UNIX e desenvolvida em estações de trabalho da SUN Microsystems (máquinas SUN-3 e SUN-4), sob o sistema operacional SUNOS, versão 4.1.1. Novos componentes foram introduzidos no sistema, que passou, então, a ser composto de (figura 1.2):

- os mesmos componentes MMS, API, APs, ACSE e INTERFACE DE OPERAÇÃO DO USUÁRIO do SISDI-MAP, mas adaptados ao ambiente UNIX;
- o protocolo ROSE;
- o protocolo RDA;
- o protocolo TP e o protocolo CCR;
- o protocolo DS (*Directory Services*), para acesso a serviços de diretório;
- o protocolo NM (*Network Management*), para a gerência da rede;

- os protocolos das camadas de Apresentação e de Sessão, agora completos, não mais apenas as suas partes básicas;
- um compilador ASN.1, responsável pela criação de rotinas de codificação e decodificação de PDUs [Res92]; e
- uma interface que mapeia os serviços do protocolo da camada de Transporte classe 4 da ISO, utilizados pelo protocolo da camada de Sessão, nos serviços do protocolo TCP (*Transmission Control Protocol*), utilizado na implementação do UNIX usada;

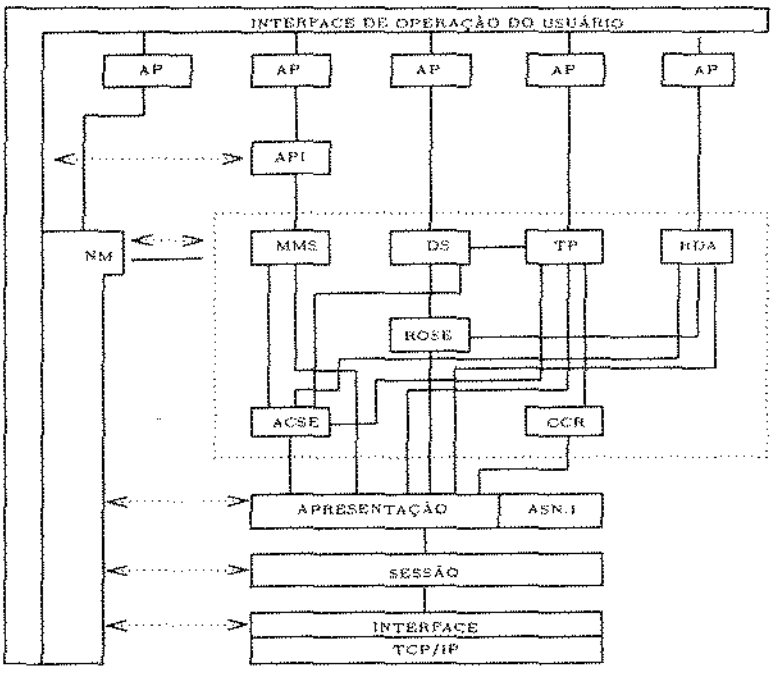

Figura 1.2: Estrutura do SISDI-OSI

Esta nova versão do sistema suportará a comunicação real entre máquinas distintas, utilizando-se os padrões da ISO para as camadas superiores e o TCP/IP para as camadas inferiores. Nesta versão, com o uso do compilador ASN.1, passa-se a tratar inclusive o problema de representação dos dados na comunicação de máquinas com diferentes representações locais. O SISDI-OSI possui a vantagem de ser um sistema completamente desenvolvido na Unicamp e, portanto, podendo sofrer quaisquer alterações e adaptações que os departamentos que o desenvolvem julgarem necessários, e serve de uma plataforma para o desenvolvimento de aplicações ou novas estruturas de comunicação que utilizem o modelo RM-OSI.

1.4 Escopo desta Tese

A camada de Aplicação do SISDI-OSI reflete o grande número de protocolos que surgiu nesta camada do RM-OSI, cada um deles fornecendo funções específicas de comunicação. O problema de

implementar os protocolos desta camada é que os relacionamentos entre eles tornaram-se complexos, como será discutido no capítulo seguinte. A ISO, em seu documento [ISO9545], fornece uma estrutura para se modelar as funções de comunicação que surgem destes relacionamentos. Porém esta estrutura é bastante abstrata.

Estes problemas devem ser tratados na implementação do SISDI-OSI. No SISDI-MAP isto não era necessário, já que possuía apenas dois protocolos de Aplicação, com relacionamentos simples entre eles. A introdução do compilador ASN.1, mencionado anteriormente, também influencia na implementação dos protocolos da Aplicação, uma vez que os dados desta camada precisam ser estruturados de maneira adequada para que as rotinas de codificação e decodificação possam atuar sobre eles.

Esta tese faz parte do desenvolvimento do SISDI-OSI e consiste das seguintes partes:

1. de uma definição de funcionalidades presentes na camada de Aplicação;

2. de um projeto da estrutura básica para implementação dos protocolos do SISDI-OSI em UNIX; e

3. da implementação do protocolo CCR, da camada de Aplicação, que segue as estruturas definidas nas partes a que os itens acima se referem.

As funcionalidades da camada de Aplicação foram determinadas a partir da análise dos conceitos do modelo RM-OSI e do funcionamento dos protocolos da Aplicação. Elas constituem um refinamento das funcionalidades definidas pela ISO. Esta parte da tese possui como objetivos esclarecer os conceitos presentes nesta camada e seus relacionamentos, e, com isto, guiar implementações. Este assunto é abordado no capítulo 2.

O projeto de implementação consiste na estruturação do sistema em termos de processos, algumas estruturas de dados e mecanismos de comunicação. O capítulo 3 trata deste problema, indicando as decisões tomadas e razões pelas quais outras alternativas foram descartadas.

O capítulo 4 descreve o protocolo CCR, como especificado nos documentos [ISO9804, ISO9805] da ISO. O capítulo 5 trata da implementação em si deste protocolo. Esta implementação foi feita em ESTELLE [ISO9074], que é uma Técnica de Descrição Formal definida pela ISO, e se utilizou o compilador para esta linguagem presente no ambiente EWS [EWS89].

O capítulo 6 apresenta as conclusões obtidas. Esta dissertação possui um anexo, que descreve as características da linguagem ESTELLE utilizadas nas implementações.

Capítulo 2

Uma Definição de Funcionalidades para a Camada de Aplicação do Modelo RM-OSI

Como mencionado no capítulo anterior, muitos protocolos têm sido definidos para a camada de Aplicação do RM-OSI. Embora os protocolos têm funções muito bem definidas, os relacionamentos entre eles se tornaram bastante complexos, devido aos tipos de funcionalidades presentes nesta camada. Entre estas funcionalidades podem ser citadas, dentre outras:

- regras de seqüenciamento envolvendo eventos de mais de um protocolo;
- a concatenação de unidades de dados de protocolo, ou, abreviadamente, PDUs, de *Protocol Data Units*, de vários protocolos. Os dados concatenados formam dados de usuário de primitivas da camada de Apresentação ou de protocolos da própria camada de Aplicação;
- a possibilidade de se agrupar subconjuntos de protocolos de Aplicação para constituir entidades de Aplicação diferentes e ter que endereçá-las;
- a obtenção de informações de endereçamento utilizando-se o protocolo DS (*Directory Services*). Este protocolo é responsável pelo fornecimento de serviços de diretório, que são serviços para se obter e armazenar informações sobre objetos do mundo real em geral, em especial os necessários à comunicação no ambiente OSI;
- a necessidade de se controlar e sincronizar os resultados em mais de uma associação.

A ISO criou uma estrutura conceitual para a modelagem destas funcionalidades em seu documento [ISO9545], mas esta é bastante abstrata. Este capítulo procura definir mais precisamente estas funcionalidades, tentando tornar mais claro o entendimento dos componentes da camada de Aplicação e seus relacionamentos. Com isto pretende-se fornecer uma visão global do funcionamento da camada, fazendo com que fique mais fácil compreender as estruturas dos protocolos da ISO, e sirva como base para a estruturação de implementações de sistemas de comunicação baseados no modelo RM-OSI. Em particular as idéias deste capítulo têm sido usadas no desenvolvimento do SISDI-OSI. As funcionalidades apresentadas para a camada, no entanto, são conceituais, fazendo com que estruturas de implementação tenham que ser derivadas a partir dela.

Nas seções 2.1 a 2.3 são descritos os conceitos da camada de Aplicação e alguns de seus procedimentos, para que se possam definir as funcionalidades para os elementos desta camada na seção 2.4. Especificamente a seção 2.1 apresenta os conceitos relacionados a esta camada que aparecem nos documentos da ISO, com alguns exemplos, de maneira a torná-los mais claros. A seção 2.2 comenta alguns aspectos dos relacionamentos entre estes conceitos, incluindo algumas interpretações. A seção 2.3.1 apresenta o esquema de endereçamento utilizado no modelo RM-OSI, no que se refere à camada de Aplicação, como informações necessárias a este esquema são obtidas, aspectos relacionados ao estabelecimento de associações e como é feito o uso de recursos fornecidos pelas camadas inferiores. Estes assuntos foram abordados pois são necessários ao funcionamento dos protocolos desta camada. A seção 2.4 apresenta, como mencionado, as funcionalidades da camada.

2.1 Definições Conceituais da Camada de Aplicação

Nesta seção, será dada uma visão geral dos conceitos relacionados à estrutura da camada de Aplicação do RM-OSI, como definida em [ISO7498] e [ISO9545]. Como uma leitura adicional, [Mel86] apresenta o relacionamento que há entre alguns conceitos e funções presentes nas camadas superiores do modelo RM-OSI e o atendimento a requisitos que aparecem nas tarefas de cooperação em sistemas distribuídos.

2.1.1 Entidade de Aplicação, Elemento de Serviço de Aplicação e Associação de Aplicação

No modelo OSI, os elementos ativos que executam as funções de uma camada são chamados de *entidades*. As entidades fornecem serviços para as entidades da camada imediatamente superior e, para isto, podem usar serviços providos por entidades da camada imediatamente inferior. Em um mesmo sistema somente entidades de camadas adjacentes se comunicam, através da troca de *primitivas*. Em sistemas diferentes, uma entidade só se comunica com outra da mesma camada. Estas entidades comunicantes em sistemas diferentes são chamadas de *entidades pares*. A comunicação é feita utilizando-se um *protocolo*, que é uma definição de um conjunto de procedimentos e de um conjunto de dados. Os procedimentos especificam exatamente quais são as ações que uma entidade deve executar ao receber eventos. O conjunto de dados é formado pelas *unidades de dados do protocolo*, ou, abreviadamente, *PDUs* , de *Protocol Data Units*, que são as unidades de informação que podem ser trocadas entre as entidades. A figura 2.1 ilustra estes conceitos.

As atividades de processamento de informação para uma determinada aplicação em um sistema real são representadas no modelo RM-OSI por um *Processo de Aplicação*, ou, simplesmente, *AP* (*Application Process*). Pode-se ter, por exemplo, APs para reservas de passagens aéreas ou para transferência de um arquivo de uma máquina para outra. Os APs estão fora do sistema de comunicação, ou seja, acima da camada de Aplicação, e são os usuários dos serviços fornecidos por esta camada. A função do RM-OSI é justamente atender aos requisitos de comunicação que surgem nestes APs.

As entidades da camada de Aplicação são chamadas de *Entidades de Aplicação*, ou, abreviadamente, *AEs*, de *Application Entities*. A um AP estão associadas uma ou mais AEs. As AEs constituem o conjunto de capacidades de comunicação do AP a que estão associadas, representando todos os aspectos deste AP que se referem a estas capacidades. Uma AE só pode estar ligada a

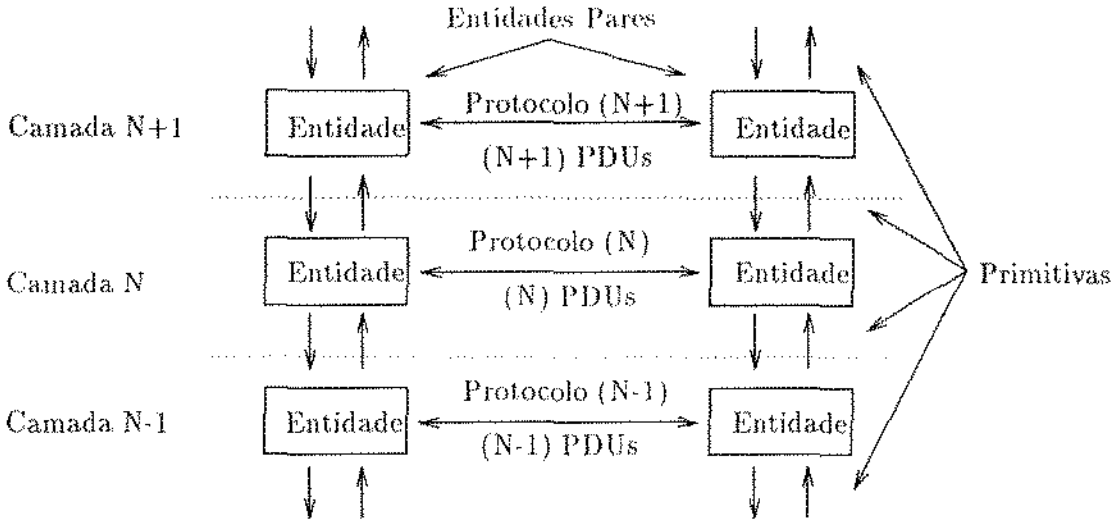

Figura 2.1: Relacionamento de comunicação entre entidades no RM-OSI

um AP. A cada AE está associado o seu tipo, chamado *Tipo da Entidade de Aplicação*, que é uma descrição das capacidades de comunicação da AE.

As AEs incorporam um conjunto de *Elementos de Serviços de Aplicação*, ou, abreviadamente, *ASEs*, de *Application Service Elements*. Os ASEs representam uma coleção de capacidades de comunicação da camada de Aplicação, para a execução de tarefas de comunicação específicas. Há ASEs que fornecem serviços para o acesso a arquivos remotos (o ASE FTAM), para acesso a bancos de dados em outros sistemas (o ASE RDA), para o estabelecimento e liberação de associações (o ASE ACSE), dentre outros. Utilizando-se de um protocolo, um ASE local e um remoto trocam informações enviando APDUs (*Application PDUs*) através de associações. Uma AE utiliza as tarefas específicas de comunicação de seus ASEs para a composição de serviços de comunicação mais genéricos.

Cada uso particular de um AP é modelado por uma *Invocação de AP* (*API*, de *Application Process Invocation*). De modo análogo, o uso das capacidades de comunicação de uma AE em uma ocasião específica é representado por uma invocação desta AE, chamada genericamente de uma *invocação de AE*, ou, abreviadamente, uma *AEI*, de *Application Entity Invocation*. Uma AEI modela o agrupamento de um sub-conjunto (ou o conjunto inteiro) das funções de comunicação da camada de Aplicação e a informação de estado relacionada a uma instância de uso destas funções por uma invocação de AP. No exemplo dado acima, de um AP representando um programa que transfere um arquivo de uma máquina para outra, cada execução deste AP para a transferência de um arquivo específico representa uma invocação deste AP. Este AP utiliza funções de comunicação, como, por exemplo, o uso de um protocolo específico para transferência de arquivos (o FTAM), que estão representadas por uma AE. Uma invocação desta AE vai representar o uso, pela API, destas funções, juntamente com informações de estado decorrentes.

A comunicação entre APIs é modelada no RM-OSI pela comunicação entre suas AEIs. Pelo modelo as APIs se interagem com AEIs locais e estas se comunicam com AEIs remotas, utilizando

protocolos. No sistema remoto, da mesma forma, AEIs se interagem com a API remota (figura 2.2).

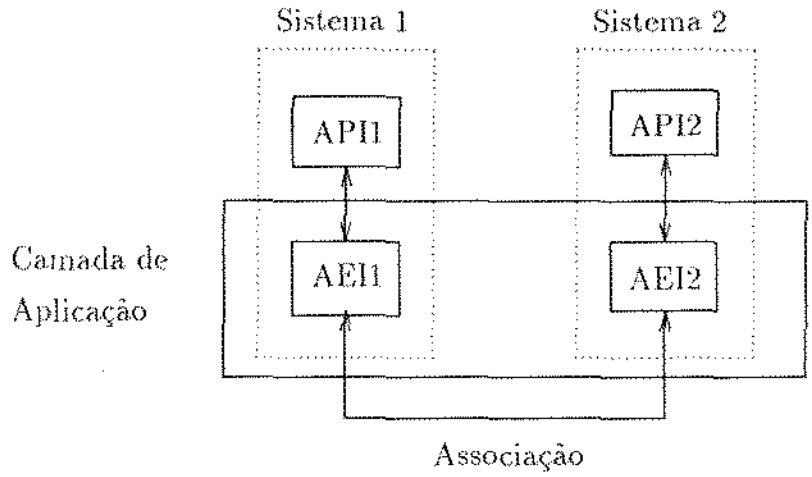

Figura 2.2: Sistemas, APIs, AEIs, Associação e Camada de Aplicação

Para os protocolos da Aplicação até agora definidos pela ISO, para que uma comunicação entre duas AEIs se processe, uma *associação de aplicação*, ou, simplesmente, *associação*, deve ser estabelecida previamente (uma associação também é mostrada na figura 2.2). Uma associação modela o relacionamento de cooperação entre AEIs, para suportar a troca de informação e a coordenação de suas atividades. Representa também um canal entre as AEIs, dentro do qual as trocas de mensagens se darão. No estabelecimento de uma associação se estabelecem quais AEIs se comunicarão, de que modo a comunicação será feita (por exemplo, negociando-se o *contexto de aplicação* a ser usado. Contextos de aplicação serão tratados na próxima seção) e valores que especificam como resursos das camadas inferiores (ver seção 2.3.4) serão usados e "inicializados".

Uma invocação de AE pode ter nenhuma, uma ou mais de uma associação em um dado momento, e as associações podem ser estabelecidas entre AEIs de AEs de tipos diferentes.

2.1.2 Contexto de Aplicação

Duas invocações de AE se comunicam através de associações. Para que esta comunicação possa ser realizada, ambas têm que estar cientes e seguirem um conjunto de regras que coordene a comunicação através desta associação. Dá-se o nome de *Contexto de Aplicação* [ISO9545] a um conjunto de tais regras. Exemplos de aspectos da comunicação a que um contexto de aplicação se refere são:

1. quais ASEs serão usados em uma associação. Cada ASE é responsável por funções de comunicação específicas e, portanto, para se processar a tarefa de comunicação em uma associação, as funcionalidades de mais de um ASE podem ser necessárias. Por exemplo, para se transferir um arquivo em uma associação, os ASEs FTAM (para a transferência em si do arquivo) e ACSE (para estabelecimento e liberação da associação) devem ser utilizados;

2. se haverá concatenação de APDUs ou não na associação;

3. quais APDUs podem ser concatenadas e em que seqüência;

4. regras para estabelecer o seqüenciamento correto de primitivas de diferentes ASEs utilizados em uma associação (estas regras estendem as regras de seqüenciamento impostas pelos ASEs individualmente); e

5. regras para o uso dos recursos de camadas inferiores, como *tokens* e pontos de sincronização. Estes recursos serão comentados na seção 2.3.4.

Uma associação tem somente um contexto de aplicação e o contexto exato a ser usado é negociado na fase de estabelecimento da associação, embora esta negociação seja limitada. Uma vez tendo-se determinado um contexto de aplicação, não se pode mais substituí-lo por outro, embora ele possa se alterar durante o tempo de vida da associação. As alterações que pode sofrer são definidas por regras presentes nele próprio.

Vários contextos de aplicação existem. Como exemplo podem ser citados os dois contextos de aplicação especificados no documento do protocolo RDA [ISO9579-1] : o *Contexto de Aplicação Básico* e o *Contexto de Aplicação TP*. No primeiro só são usados os ASEs ACSE e RDA. Neste contexto o ASE ACSE trata do estabelecimento e liberação da associação e o ASE RDA cuida dos mecanismos para acesso ao banco de dados remoto, como também do gerenciamento de diálogos e transações. No segundo, são usados os ASEs ACSE, RDA, TP e, opcionalmente, o CCR. Neste contexto o RDA passa a utilizar os mecanismos de gerenciamento de diálogos e transações do TP. Estes contextos de aplicação, embora sejam usados para o mesmo objetivo geral, ou seja, acessar bancos de dados remotos, diferenciam-se em algumas funcionalidades, como, por exemplo, mapeamentos de APDUs em primitivas da Apresentação e regras de seqüenciamento de primitivas. Além disto com o primeiro contexto a transação só pode englobar dois sistemas. Com o segundo, inúmeros sistemas podem fazer parte de uma transação. Um contexto de aplicação possui, como sentido geral, indicar que tipos de serviços se pode obter através de uma associação.

2.1.3 SAO, SACF e MACF

A figura 2.3 mostra como uma invocação de AE é internamente estruturada [ISO9545]. Nesta figura:

1. há um objeto chamado *SAO* (*Single Association Object*, Objeto de uma Única Associação) acoplado a cada associação. Este objeto modela as funções de controle e informações de estado relacionadas àquela associação. Uma invocação de AE pode ter nenhum, um ou mais de um SAO em um dado instante.

   Cada SAO é dividido em:

   - um conjunto dos ASEs que serão usados na associação. O ACSE é sempre um elemento deste conjunto, porque é o ASE que é responsável pelo estabelecimento e liberação das associações e é necessário que duas AEIs estabeleçam uma associação antes de trocarem informações; e

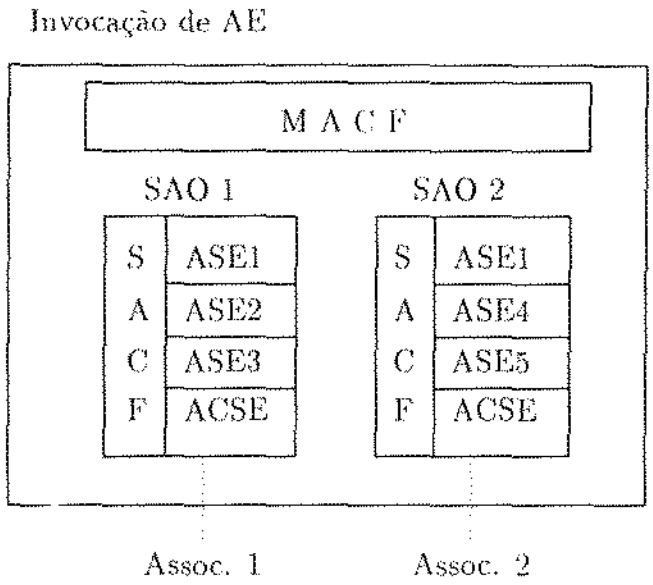

Figura 2.3: Invocação de AE com MACF, SAOs, SACFs, ASEs e associações

- o SACF (*Single Association Control Function*, Função de Controle de uma Única Associação), que é a parte do SAO responsável pelo controle das interações entre os ASEs que participam da associação e do uso dos recursos das camadas inferiores;

2. e há um outro componente, chamado MACF (*Multiple Association Control Function*, Função de Controle de Múltiplas Associações), que modela as funções necessárias ao controle das atividades que envolvem mais de uma associação.

2.2 Interpretação dos Relacionamentos entre os Conceitos

Nesta seção serão comentados alguns aspectos de algumas das relações entre os conceitos discutidos na seção anterior.

A um AP pode estar associada uma ou mais de uma AE, cada uma sendo de um tipo diferente. Por [ISO9545] é possível ter duas AEs do mesmo tipo associadas a um AP. Esta situação não é considerada aqui, porque a associação de AEs a APs serve para indicar as capacidades de comunicação que o AP utiliza. Estas funções são as fornecidas pelas AEs associadas a ele (AEs modelam agrupamentos de capacidades de comunicação da camada de Aplicação). Desta forma não é necessário haver duas AEs de um mesmo tipo associadas a um AP, já que uma é o bastante para que as capacidades de comunicação que ela representa sejam atribuídas ao AP.

Uma invocação de AP, por sua vez, pode estar associada a muitas invocações de AE de um mesmo tipo, já que este tipo de associação representa ocasiões de uso das capacidades de comunicação da AE pelo AP. A invocação de AE modela a união de funções de comunicação e de controle e informações de estado relacionadas a uma destas ocasiões de uso. Esta informação de estado modela o efeito global resultante das atividades executadas em todas as associações desta AEI. Desta

maneira pode-se modelar, por exemplo, associações que se interrompem no tempo, sendo restauradas posteriormente [ISO9545]. A AEI representa também a coordenação das atividades em suas associações, que podem ser simultâneas ou consecutivas.

As AEs servem para descrever as capacidades de um sistema de comunicação, formadas por capacidades básicas (os ASEs). Por exemplo, não basta saber que um sistema possui implementado os protocolos TP e RDA para poder usá-los em uma única transação. É preciso saber se há uma AE que permita usá-los em conjunto, formando uma tal transação. As AEs de um sistema deveriam modelar os tipos de inter-relacionamentos possíveis entre os protocolos da camada de Aplicação que o sistema suporte. Desta maneira não deveria haver uma AE que suportasse tanto associações para transferência de arquivo (FTAM) e acesso a bancos de dados (RDA) se estas associações não se relacionarem. Não há inter-relacionamento a ser modelado. A estruturação de AEs segundo estas idéias seria modular. Informações sobre AEs de um sistema podem ser obtidas acessando-se diretórios de informações de comunicação. O protocolo DS é utilizado para este fim [CCITT500].

Observando o que uma AEI modela dever-se-ia concluir que as atividades executadas nas associações desta AEI deveriam ter alguma relação entre si e que esta relação fosse em termos de informação de estado na própria AEI, ou seja, que o estado de uma associação deveria depender do estado de um sub-conjunto das outras associações e que esta dependência fosse em termos de aspectos que a própria AEI deva tratar, não o AP. Uma AEI não deveria conter uma associação que não se relacione com as demais, pois, neste caso, não haveria informação de estado ou controle a ser modelado entre estas associações.

Para ilustrar, suponha que um AP tenha que ler dois arquivos, cada um em um sistema remoto diferente, e deva concatenar os conteúdos destes arquivos. Neste exemplo há um AP associado a apenas uma AE. Esta AE forneceria as funções de comunicação necessárias para ler o arquivo (especificamente, esta AE conteria o ASE FTAM). Seguindo a interpretação acima, a invocação de AP estaria associada a duas invocações de AE, cada uma com apenas um SAO. Cada SAO suportaria a associação com um sistema remoto (figura 2.4a). Desta maneira ficaria excluída uma modelagem na qual haveria uma invocação de AE com dois SAOs, um para cada associação (figura 2.4b). A primeira opção é preferível porque as associações conceitualmente só têm *dependências de estado* no AP, ou seja, as associações só se relacionam em aspectos que devem ser tratados pelo AP, portanto, fora da camada de Aplicação.

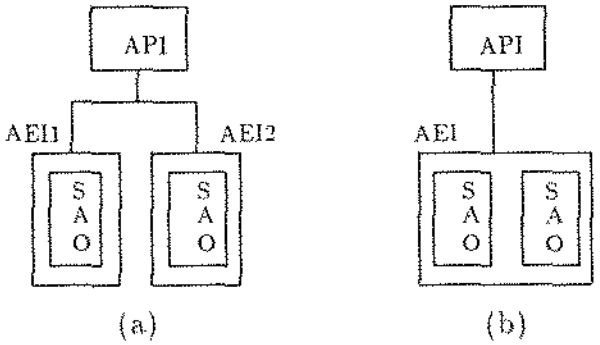

Figura 2.4: Relacionamento entre APIs, AEIs e SAOs

O RM-OSI, no entanto, não define precisamente que tipos de relações de estado e coordenação deve haver entre componentes de uma AEI. Permite, com isto, mais flexibilidade na constituição das AEIs, possibilitando inclusive a situação mostrada na figura 2.4b. Porém a definição de entidade que aparece em [ISO7498-3][1] e que, segundo uma nota neste documento, passará a ser a definitiva em uma revisão de [ISO7498], parece sugerir que as AEs devam fornecer um serviço de comunicação composto coerente, modular, seguindo, assim, as idéias descritas nos parágrafos anteriores.

Um outro exemplo de relacionamento entre associações de uma AEI que aparece em alguns protocolos é que, para fins de ganho em eficiência, uma AEI pode conter uma série de associações com outras AEIs e gerenciá-las, de maneira que possam ser usadas para suportar várias seções de transmissões, sem ter que liberá-las e novamente estabelecê-las em cada uma destas seções. Com isto pode-se ter, por exemplo, algo como a figura 2.5, onde a AEI pode manter várias associações ativas para uso futuro. Esta idéia é a utilizada nos *pools* de associações dos protocolos TP [ISO10026-3] e RDA [ISO9579-1], que representam exatamente conjuntos de associações deste tipo. Uma AEI deste tipo poderá ter vários MACFs, independentes, cada um atuando sobre o sub-conjunto dos SAOs que estiverem suportando associações relacionadas umas com as outras. Mas, mesmo neste caso, o tipo de serviço fornecido pela AE deveria manter-se coerente.

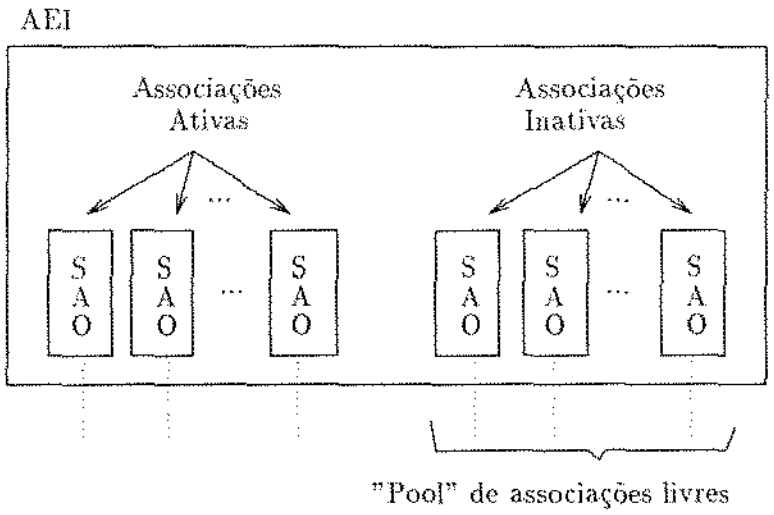

Figura 2.5: *Pool* de associações livres

Da mesma maneira que o contexto de aplicação define regras para o controle de uma associação, executadas pelo SACF, poderia haver um conceito análogo para o MACF. No entanto isto não ocorre porque os requisitos de comunicação que possam surgir nos MACFs têm que ser atendidos pelos contextos de aplicação das associações, pois é através delas que as AEIs se comunicam. Portanto pode haver regras nos contextos de aplicação que se aplicam ao funcionamento de MACFs. Por exemplo, podem ser definidas regras para a sincronização de funções de coordenação distribuídas, como é o caso da ordem de efetivação de uma transação no TP [ISO10026-3]. Uma vez decidida

[1] Neste documento define-se uma entidade de uma camada *N* como sendo : *um elemento ativo de um sub-sistema N, que incorpora um conjunto de capacidades definidas para a camada N, e que corresponde a um tipo de entidade N específico (sem quaisquer capacidades extras sendo usadas)*

pela raiz da árvore de transação, a indicação de efetivação deve ser comunicada aos demais nós da árvore. Cada nó possui um MACF que controla as associações estabelecidas com seus subordinados, através das quais esta informação deve ser re-transmitida.

2.3 Características de Procedimentos Comuns aos Contextos de Aplicação

Nesta seção serão discutidos aspectos de endereçamento, estabelecimento de associação e uso de recursos fornecidos pelas camadas inferiores, pois são funcionalidades que influem no modo de se estruturar a camada de Aplicação e que são usadas por todos os contextos de aplicação.

2.3.1 Endereçamento na Camada de Aplicação

O RM-OSI, em seu documento [ISO7498-3], define *nomes* como sendo construções lingüísticas que correspondem a um objeto em um universo de discurso; *título* como sendo um nome associado a um objeto quando o propósito do nome é distinguir este objeto de outros; e *identificadores* como sendo nomes usados para distinguir diferentes instâncias de um objeto. Por isto títulos são usados para nomear APs e AEs, resultando em *títulos de AP* e *títulos de AE*, enquanto identificadores são usados para nomear invocações de AP e de AE, tendo-se com isto *identificadores de invocações de AP* e *identificadores de invocações de AE*.

Para serem endereçadas, as entidades das camadas se ligam a *SAPs* (*Service Access Points* - Pontos de Acesso a Serviços). Estes SAPs representam, conceitualmente, os pontos nas interfaces entre as camadas onde as entidades destas camadas trocarão primitivas de serviços. Os SAPs recebem designações em seus nomes para identificar a interface na qual se encontram. Especificamente recebe uma abreviação do nome da camada que está abaixo da interface. Com isto se tem, por exemplo: *Presentation-SAPs* (SAPs da Apresentação), abreviados como P-SAPs, referindo-se a SAPs entre a camada de Aplicação e a de Apresentação e é por onde as entidades de Aplicação enviam e recebem primitivas de serviços das entidades da camada de Apresentação; e *Session-SAPs* (SAPs da Sessão), abreviados como S-SAPs, que ficam entre a camada de Apresentação e a de Sessão e são os pontos através dos quais entidades de Apresentação e Sessão trocam primitivas de serviços da camada de Sessão. A figura 2.6 ilustra estes conceitos. Como SAPs estão em interfaces entre camadas, não há SAPs acima da camada de Aplicação, nem abaixo da camada Física.

Existem regras para o relacionamento entre SAPs e entidades [ISO7498]. Para este capítulo é de interesse apenas como estas regras se aplicam a P-SAPs e AEs, pois são os SAPs e entidades que se relacionam com a camada de Aplicação. Para eles as regras reduzem-se a cada P-SAP estar ligado a no máximo uma AE e a uma entidade de Apresentação. As AEs, por sua vez, podem estar ligadas a mais de um P-SAP. Estas regras fazem com que cada P-SAP identifique a entidade de Aplicação e a de Apresentação a que está ligada. Este relacionamento é dinâmico, ou seja, pode variar no tempo. Em cada instante de tempo há um *endereço de Apresentação* associado a cada título de AE. Este endereço de Apresentação identifica o conjunto de P-SAPs ligados à AE (figura 2.7).

Note que, uma vez endereçado um SAP, as entidades à qual este SAP está ligado ficam também esdereçadas. Nisto se baseia o esquema de endereçamento utilizado no RM-OSI. Para se endereçar um AP, basta que se tenha o endereço dos P-SAPs ao qual uma AE deste AP está ligada, pois,

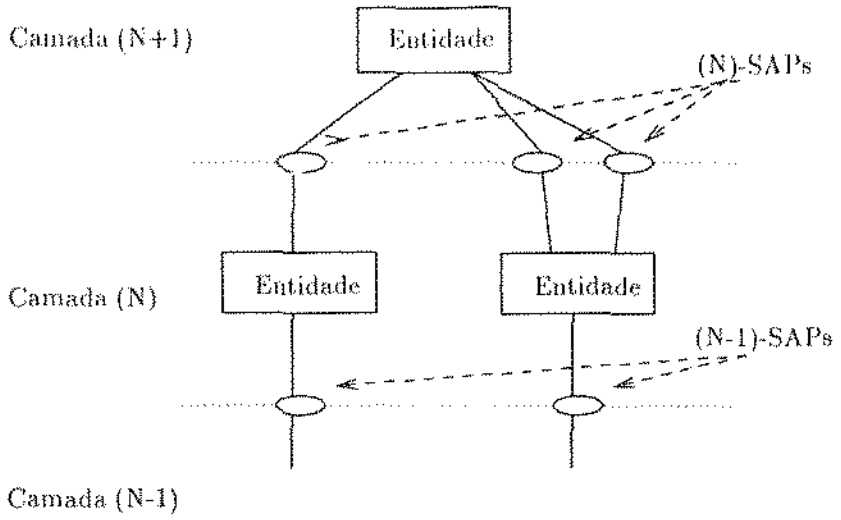

Figura 2.6: Relação entre SAPs e Camadas

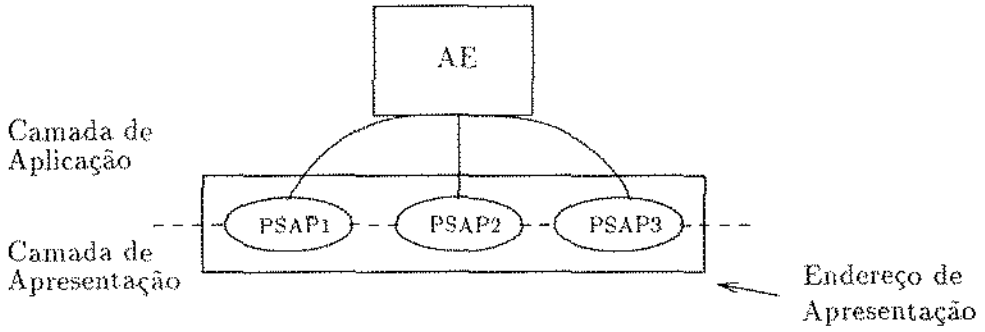

Figura 2.7: Relacionamento entre AE, PSAPs e Endereço de Apresentação

com isto, determina-se exatamente a AE (pela regra de relacionamento entre P-SAPs e AEs) e daí se acessa o AP (uma AE está ligada a um único AP).

O endereço de Apresentação é usado para endereçar AEs, não invocações de AEs. Na fase de estabelecimento de uma associação, este endereço tem que ser obrigatoriamente fornecido pela AEI que está requisitando o estabelecimento. Opcionalmente ela pode fornecer também um identificador de invocação de AP e/ou um identificador de invocação de AE, para especificar os objetos exatos com os quais se quer comunicar. Pode ser enviado também um título de AE ou título de AP, para se verificar se a AE e o AP com que se deseja comunicar são os que realmente são acessados através dos P-SAPs identificados pelo endereço de Apresentação. Esta verificação pode ser necessária pois, como mencionado, as ligações de AEs com P-SAPs são dinâmicas e endereços de apresentação podem-se tornar inválidos. O uso destes títulos permite especificar a AE e o AP independentemente do endereço de apresentação. Informações análogas sobre o sistema que está requisitando o

estabelecimento da associação são também enviadas à entidade remota, para o endereçamento no sentido de comunicação inverso.

O esquema de endereçamento assume a existência de duas *Facilidades de Diretório*, que representam provedores de informação de endereçamento [ISO7498-3]. Para a camada de Aplicação há uma destas facilidades, chamada a *Facilidade de Diretório de Títulos da Aplicação*, que, a partir de títulos de APs e AEs, fornece informações de endereçamento utilizadas nas camadas superiores. A outra facilidade é usada na camada de Rede e, a partir de endereços de N-SAPs (*Network-SAPs* – SAPs que estão entre a camada de Rede e a de Transporte), fornece informações de endereçamento adicionais usadas nas camadas inferiores. Neste texto só é de interesse a primeira destas facilidades, que influencia no comportamento dos protocolos da camada de Aplicação.

Os títulos usados para se obter informações da *Facilidade de Diretório de Títulos de Aplicação* podem ser genéricos ou não, ou seja, especificam um conjunto de APs ou AEs, no primeiro caso, ou apenas um AP ou uma AE, no segundo. Os títulos genéricos são títulos de *tipos* de APs e AEs. Títulos não genéricos são os títulos de APs e AEs. Se esta facilidade for consultada fornecendo-se um título genérico, a facilidade retornará uma lista de títulos não genéricos. Se títulos não genéricos forem fornecidos, as informações de endereçamento serão retornadas.

No RM-OSI as funções de uma camada que tratam de endereçamento são chamadas de *Funções de Diretório*. Estas funções processam endereços de uma camada, endereços da camada inferior, títulos de entidades da camada, informações de endereçamento vindas em PDUs e fazem mapeamentos entre estas categorias de informação, podendo usar, em certos casos, informações locais e de roteamento. Estas funções são realizadas pelas entidades da camada e são executadas ou no momento de estabelecimento de uma conexão ou, em protocolos não orientados a conexão, durante cada transmissão de dados. Na camada de Aplicação, portanto, estas funções são realizadas pelas AEs e no momento de estabelecimento de uma associação.

Nesta camada as funções de diretório irão fazer mapeamentos apenas entre títulos de entidades da Aplicação e endereços de Apresentação, pois as outras categorias de informação manipuladas por funções de diretório não se aplicam a esta camada. Especificamente, as funções que cada AE terá que executar são [ISO7498-3]:

1. no sistema que solicitou o estabelecimento de uma associação:

   função 1 a partir do título da entidade de aplicação com que se quer comunicar e de quaisquer informações adicionais que forem necessárias, obtém o endereço de apresentação desta entidade remota (que será passado em uma primitiva de requisição de um serviço de Apresentação). Estas informações adicionais podem ser usadas na seleção de um título de AE não genérico, a partir da lista fornecida pela Facilidade de Diretório de Títulos de Aplicação;

   função 2 a partir do endereço de apresentação da entidade remota e de quaisquer informações adicionais que forem necessárias, obtém o endereço de apresentação da entidade local (que será passado em uma primitiva de requisição de um serviço da Apresentação) e o P-SAP exato no qual se enviará a primitiva;

2. no sistema respondedor:

   função 3 ao receber uma primitiva de indicação de um serviço da Apresentação, a partir do endereço de apresentação chamado e de quaisquer informações adicionais que forem

necessárias, retorna o endereço de apresentação de resposta. Este endereço pode se diferir do endereço chamado, pois as informações de endereçamento que um sistema possui sobre as entidades de um outro podem deixar de ser válidas. O sistema receptor pode, ao receber um endereço antigo inválido, enviar um novo como endereço de resposta.

Estas funções interagem com mecanismos locais para obtenção das informações necessárias. Se as informações estiverem em sistemas remotos, o protocolo DS [CCITT500] é usado. O modo como é utilizado é comentado na seção seguinte.

2.3.2 Uso dos Serviços de Diretório (DS) para o Suporte a Endereçamento

O protocolo DS [CCITT500] é o utilizado para acessar serviços de diretório em um sentido genérico, ou seja, serviços que permitem a obtenção e armazenamento de informações sobre objetos do mundo real. No ambiente OSI é utilizado para se obter informações que facilitam a comunicação entre sistemas, por exemplo, informações sobre APs, AEs, listas de distribuição de mensagens, etc.. Em particular, pode ser utilizado para se obter endereços de apresentação associados a AEs. Este uso específico é importante pois está relacionado a endereçamento.

Para o estabelecimento de uma associação, o endereço de Apresentação da AE remota com que se quer comunicar é uma informação obrigatória. De acordo com as especificações dos protocolos de Aplicação a obtenção destas informações pode ser feita ou pelos APs, como, por exemplo, no caso do FTAM [ISO8571], ou por algum elemento interno à AE, como no TP [ISO10026-2, ISO10026-3]. Neste segundo caso, o AP deve fornecer certas informações, pois são os usuários que especificam com quem efetivamente se deseja comunicar. Poderiam, por exemplo, fornecer o título do AP ou da AE com que desejam estabelecer uma associação, deixando para o sistema de comunicação (através da AE) a tarefa de obter os endereços de Apresentação adequados.

Estas informações podem ser conseguidas pelo acesso a uma Facilidade de Diretório, por algum acordo feito fora do sistema de comunicação (por exemplo, usando sempre um endereço para acessar a AE remota) ou por informações obtidas de comunicações feitas anteriormente [ISO7498-3].

No caso dos endereços de apresentação serem obtidos pelo AP, este pode fazê-lo de dois modos:

1. se as informações estão disponíveis localmente (no mesmo sistema), ele interage com a facilidade de diretório, ao nível de processo de aplicação, para obtê-las. Como esta interação é feita é um problema de implementação local, e poderia ser feita, por exemplo, usando-se um outro AP (figura 2.8a). Este AP seria responsável pelo acesso às informações do diretório;

2. se as informações estão em um sistema remoto, o AP usuário ou interage com um outro AP no mesmo sistema, que, de modo análogo ao item anterior, seria responsável pela aquisição das informações nos diretórios locais ou remotos (figura 2.8b), ou ele teria uma AE que suporte o contexto de aplicação *Acesso ao Diretório* [CCITT500] do protocolo DS, através do qual ele teria acesso aos diretórios remotos (figura 2.8c). Observe que na figura 2.8b o AP que acessa o diretório remoto também precisa de uma AE com este contexto de Aplicação.

No caso do endereço de Apresentação ser obtido por um elemento da AE e haver parte do diretório no sistema, o elemento da AE teria que interagir com um AP responsável pelo acesso às facilidades de diretório. Isto é necessário porque, segundo o modelo definido pelo DS, a base de informação do diretório é gerenciada por um AP [CCITT500], pois a tarefa de armazenamento e

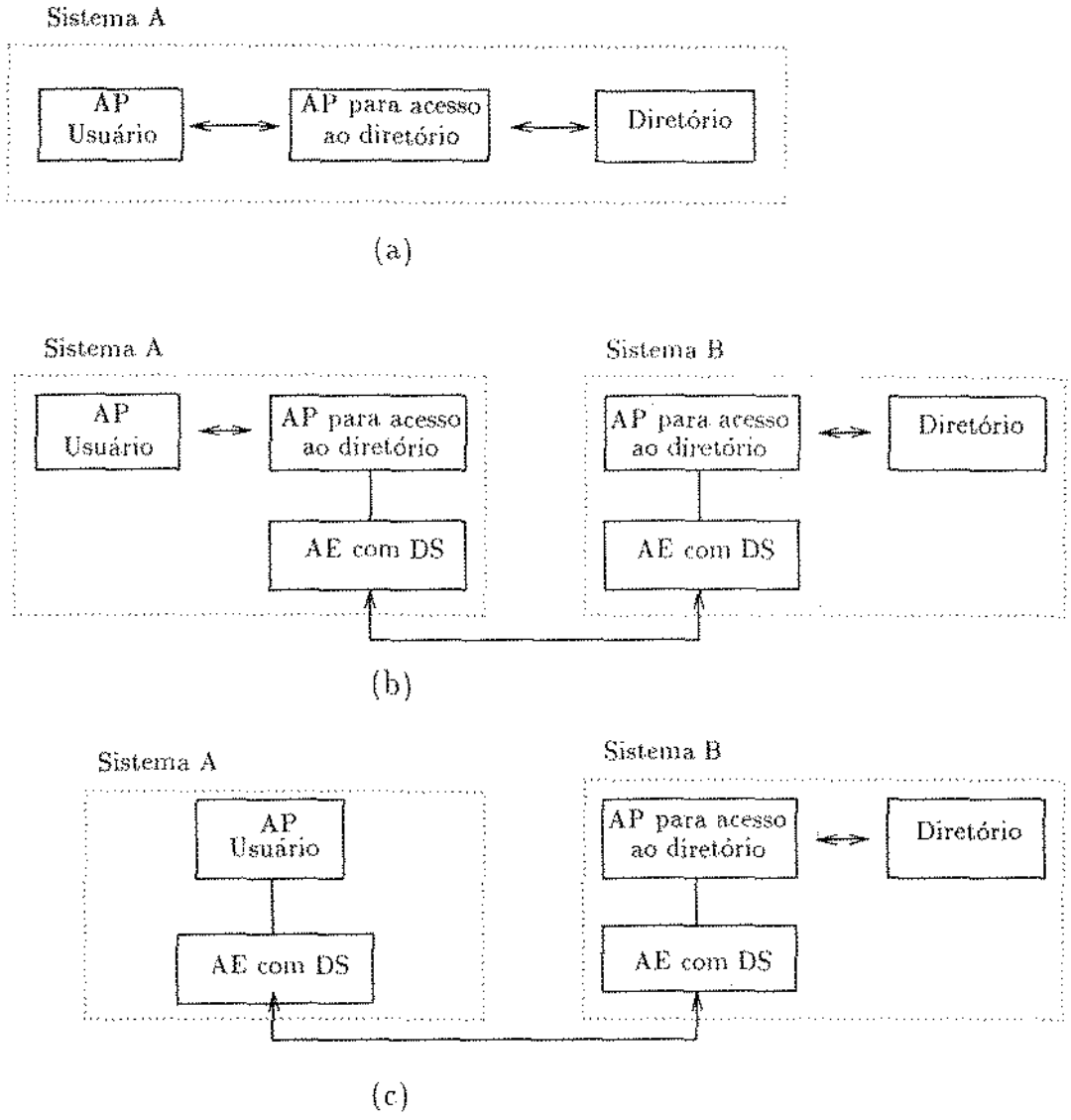

Figura 2.8: Usos do protocolo DS para obtenção de informações de endereçamento

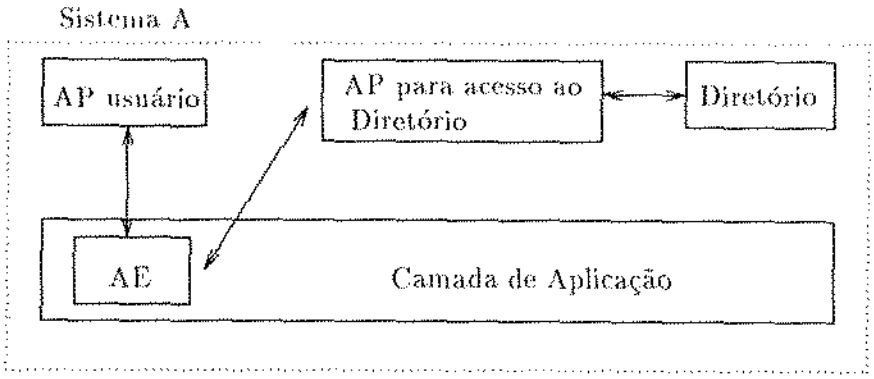

Figura 2.9: Invocação de Serviço de um AP por uma AE

acesso local aos dados do diretório está relacionada apenas a processamento de informação, devendo portanto ficar fora do sistema de comunicação. A figura 2.9 ilustra este relacionamento (nesta figura, o AP que acessa o diretório poderia também usar o DS para acessar diretórios remotos, como na figura 2.8). Este tipo de interação, ou seja, uma AE requisitando serviços a um AP, não está de acordo com o esquema de interação entre elementos definidos no RM-OSI. Segundo este modelo, AEs fornecem serviços ao AP a que estão ligadas. No entanto esta é a maneira que se pode imaginar como sendo o comportamento dos protocolos deste tipo, a partir do exame de suas especificações. [ISO7498-3] e [ISO9545] especificam que estas informações seriam fornecidas por funções de gerenciamento locais, mas estas funções simplesmente representam este relacionamento.

Caso não haja parte do diretório no sistema, o endereço de apresentação poderia ser obtido através de uma associação dentro da própria AEI que necessita de informação, contendo o contexto de aplicação *Acesso ao Diretório* do DS [CCITT500], para a comunicação com um sistema remoto que forneça as informações. Este esquema de uso não se encaixa exatamente no modelo de uso previsto em [CCITT500], por não haver um AP como usuário dos serviços de diretório e, sim, parte da AE.

A obtenção dos endereços de apresentação a partir de títulos de APs e de AEs é uma função que é auxiliar à função dos protocolos. Esta tarefa poderia, então, ser executada pelos APs, introduzindo uma funcionalidade que se adequa ao conceito de *Application Program Interface* [Mad91] do projeto MAP/TOP. Uma das funções de uma interface deste tipo é simplificar o uso dos protocolos da Aplicação pelos APs. Desta maneira o modelo ficaria mais modular (os protocolos não precisariam mais fazer estas tarefas auxiliares) e o tipo de interação na qual uma AE solicita serviços a um AP seria evitada.

Fornecendo informações de endereçamento, o protocolo DS dá suporte, na camada de Aplicação, à primeira das funções de diretório listadas anteriormente.

2.3.3 Estabelecimento de Associação

O processo de estabelecimento de uma associação não é um procedimento simples. Em termos gerais, o sistema que requer o estabelecimento, utilizando o protocolo ACSE, forma uma APDU deste

protocolo[2] e a envia usando uma primitiva do serviço P-CONNECT da camada de Apresentação [CCITT216] (associações possuem uma relação de um para um com conexões de Apresentação). No sistema destino, com o qual se quer estabelecer a associação, uma de suas AEs é endereçada pelo endereço de apresentação, informações vindas nos campos da APDU enviada e nos parâmetros da primitiva na qual chegou são analisados e, se tiverem valores viáveis, a associação é estabelecida e uma confirmação positiva retorna ao sistema original. Caso haja valores não aceitáveis, a associação não é estabelecida e uma confirmação negativa é devolvida.

Algumas características deste procedimento, de interesse para a definição de funcionalidades para a camada de Aplicação, são:

1. durante o estabelecimento da associação, há uma negociação do contexto de aplicação a ser usado;

2. ele envolve a manipulação de informações dependentes dos seguintes elementos:

   (a) da API associada à AEI na qual a associação está sendo estabelecida. Exemplos deste tipo de informação são os identificadores da API e da AEI com os quais a associação é para ser estabelecida;

   (b) do contexto de aplicação que será usado. Por exemplo, quais devem ser os valores iniciais para pontos de sincronização, ou como deve ser a posse inicial de *tokens* da Apresentação;

   (c) da AE como um todo. Por exemplo, quais contextos de aplicação a AE suporta, a qual AP a AE está ligada, quais identificadores de invocação de AE são válidos, dentre outros;

   (d) das camadas de Apresentação e Sessão, pois depende de quais unidades funcionais estão disponíveis para serem utilizadas em uma associação;

   (e) da camada de Transporte, pois depende da qualidade de serviço que pode fornecer;

3. APDUs de protocolos podem ser enviadas como dados de usuário da primitiva que estabelece a associação (A-ASSOCIATE.request) [CCITT217]; e

4. pode ser necessário utilizar uma outra associação da mesma AEI para o estabelecimento, como comentado na seção anterior.

O item 2b indica que o estabelecimento de associação implica em controle sobre recursos oferecidos através da camada de Apresentação, enquanto o item 3 implica em que deve haver um controle sobre as interações entre os ASEs que serão utilizados. Estas duas funcionalidades são as atribuídas a SACFs.

O item 4, no entanto, indica que este procedimento pode implicar em relacionamentos entre duas associações, o que é uma função de MACF. Os itens 2a, 2c, 2d e 2e indicam que não bastam apenas informações que se referem unicamente a um SAO ou a uma AEI.

O item 1 informa que há regras necessárias ao procedimento de estabelecimento de associação que são à parte do contexto de aplicação que for ser estabelecido. Estas regras seriam aquelas que definem o comportamento dos componentes da AEI para o estabelecimento em si da associação,

[2]Somente em um caso, o do protocolo MHS, para processamento de correio eletrônico, uma APDU do ACSE não é enviada [CCITT227]. Mas o resto do procedimento é análogo.

até que se negocie o contexto de aplicação a ser usado. Estas regras incluem a interação com o ASE ACSE.

Em resumo conclui-se que o procedimento de estabelecimento de associação inclui funções de SACF, MACF, além de tratar informações que não se restringem a uma única AEI. Esta conclusão será utilizada posteriormente para situar esta funcionalidade na estrutura da camada de Aplicação.

2.3.4 Uso dos Recursos das Camadas Inferiores

Nesta seção se comenta sobre como os recursos das camadas inferiores são tratados na camada de Aplicação. Está-se dando o nome aqui de *recursos* a unidades funcionais, atividades, unidades de diálogo, *tokens* e pontos de sincronização da camada de Sessão, unidades funcionais e contextos de apresentação da camada de Apresentação e qualidade de serviço da camada de Transporte. Estes são recursos presentes nas camadas inferiores que representam facilidades oferecidas por estas camadas, mas que é na camada de Aplicação que são selecionados ou controlados. Todos estes recursos são fornecidos à camada de Aplicação através da camada de Apresentação, que é a única a que a primeira tem acesso.

As unidades funcionais das camadas de Apresentação e Sessão necessárias a um contexto de aplicação estão definidas neste próprio contexto, e são selecionadas no momento de estabelecimento das conexões das respectivas camadas. Como estas conexões possuem uma relação de um para um com associações, este momento coincide com a fase de estabelecimento destas. Parâmetros das primitivas do ASE ACSE para o estabelecimento das associações permitem que os usuários destas primitivas forneçam os nomes das unidades funcionais de que necessitam. É tarefa dos provedores de serviços de Apresentação e Sessão aceitarem ou não o estabelecimento das conexões em função de se suportam ou não as unidades funcionais requisitadas. Ao se estabelecer uma associação, o usuário do serviço A-ASSOCIATE do ACSE sugere as unidades funcionais especificadas no contexto de aplicação que se estiver negociando e controla os resultados fornecidos pela entidade par e pelas entidades das camadas inferiores. Informações sobre as unidades funcionais utilizadas nas camadas têm que ser mantidas na camada de Aplicação. A razão disto se tornará mais clara posteriormente.

A camada de sessão permite a estruturação do diálogo realizado através de uma associação em partes lógicas e que se estabeleçam pontos de sincronização na transmissão dos dados. O diálogo é dividido em unidades chamadas *unidades de diálogo*. Unidades de diálogo agrupadas formam *atividades*. Esta estruturação permite, por exemplo, que a transferência do conteúdo de um conjunto de livros seja dividida em livros, que constituiriam as atividades, e que cada livro seja dividido em capítulos, que constituiriam as unidades de diálogo. Cada página de um capítulo poderia ser marcada por um ponto de sincronização menor. Na ocorrência de uma falha na transmissão, os sistemas poderiam se ressincronizar em, por exemplo, um capítulo anterior, de forma que o conteúdo do livro perdido com a falha fosse novamente transmitido.

Os pontos de sincronização fornecidos são de dois tipos: os maiores e os menores. Os primeiros delimitam as unidades de diálogo e as atividades; os últimos servem apenas para estabelecer pontos de sincronização dentro de uma unidade de diálogo. Na ocorrência de algum erro, a transmissão pode ser restabelecida ao estado que tinha em um destes pontos. A diferença que há entre os pontos maiores e os menores é que, uma vez estabelecido um ponto de sincronização maior, o diálogo não pode mais ser restabelecido a um ponto anterior na comunicação. Os pontos menores não impõem tal restrição.

A camada de Sessão, no entanto, apenas fornece mecanismos, através dos serviços da camada

de Apresentação, para que estes pontos e atividades sejam estabelecidos. Quem atribui semântica a eles são os componentes da camada de Aplicação que os estabelecem. Esta semântica definiria como o diálogo será dividido em atividades e unidades de diálogo e o que cada ponto de sincronização representa. Informações sobre estes recursos têm que ser mantidas na camada de Aplicação, de maneira que se possa restabelecer o estado da associação quando uma ressincronização a um dos pontos for requisitada.

Pode ser o caso de mais de um ASE fazer uso destes pontos de sincronização para a realização de alguma tarefa sua. Portanto deve haver um mecanismo de controle do uso destes recursos. O contexto de aplicação usado na associação deve conter regras que regularizem este uso e tem que haver mecanismos no SACF que efetivem estas regras. Por exemplo o controle dos números seqüenciais dos pontos deve ser feito pelo SACF e deve ser ele também que deve armazenar informações sobre o estado da comunicação nos diversos pontos. A estruturação do diálogo em atividades e unidades de diálogo também deve estar definida no contexto de aplicação, de maneira que também não haja incompatibilidade na definição.

A camada de Sessão define também *tokens* que, em termos gerais, atribuem capacidades à entidade que está de posse deles. Por exemplo um *token* de dados só permite que a transmissão de dados seja feita pela AEI que o possuir. Este é o modo de se estabelecer um diálogo *half-duplex* entre dois sistemas. Uma situação análoga ao caso do parágrafo anterior ocorre também com os *tokens*, ou seja, mais de um ASE pode usá-los. Da mesma maneira regras para o controle deste uso devem ser estabelecidas no contexto de aplicação.

Os *tokens* podem também ser usados para evitar certas situações de conflito de mensagens durante as transmissões. Regras podem ser impostas, por exemplo, para certos serviços só serem invocados se a entidade estiver de posse de um determinado *token*. Todos estes controles devem ser realizados pelo SACF.

Contextos de apresentação são pares de sintaxes abstratas e sintaxes de transferência. Estes contextos são usados para transmitir APDUs, que podem ser formados por dados cujas estruturas são definidas em sintaxes abstratas mencionadas no contexto de aplicação e dados que são definidos pelos usuários. Durante o tempo de vida de uma conexão de apresentação há associado a ela um conjunto de contextos de apresentação ativos. Este conjunto pode alterar-se, recebendo novos contextos ou eliminando-se outros. Os contextos utilizados para transmitir os primeiros tipos de dados são especificados no momento de estabelecimento da associação. Os contextos para os outros podem ser especificados também nesta fase ou durante a execução das atividades sobre a associação. Neste caso o usuário informa que alteração quer efetuar no conjunto de contextos vigente e é função do SACF interagir com a camada de Apresentação para este fim. O conjunto de contextos de apresentação para uma associação vai sendo atualizado de acordo com as necessidades de transmissão de APDUs.

Qualidade de serviço é uma especificação do tipo de serviço desejado na associação. Nesta qualidade se definem características como tempo de estabelecimento da associação, probabilidade de falha neste estabelecimento, probabilidade de falha na transmissão de dados, dentre outros. Este parâmetro é definido na camada de Aplicação ao se usar o serviço A-ASSOCIATE do ACSE. Este valor pode ser estabelecido no contexto de aplicação que se estiver negociando para a associação, por decisões locais de implementação de um sistema ou fornecidas pelos APs, através de primitivas de serviços de ASEs. Este valor vai sendo transferido através de primitivas de estabelecimento de conexões de apresentação e de sessão, até à camada de Transporte. Esta camada tenta atender à solicitação utilizando os recursos disponíveis na rede. O que é de interesse neste procedimento

é que informações sobre a qualidade de serviço que está disponível em uma associação têm que ser mantidas pela camada de Aplicação, de maneira análoga ao caso das unidades funcionais das camadas inferiores. O motivo disto se tornará claro posteriormente.

2.4 Estrutura dos Componentes da Camada de Aplicação

Nesta seção será mostrada uma definição de funcionalidades presentes na camada de Aplicação. Estas funcionalidades formam uma estrutura conceitual para os componentes desta camada que procura atender aos objetivos de esclarecer os conceitos envolvidos, especificando mais detalhadamente o que cada componente modela e os relacionamentos entre eles, e fornecer uma visão global do funcionamento desta camada. Desta maneira espera-se formar uma base conceitual mais precisa sobre a camada, na qual implementações possam se basear.

Esta estrutura é baseada na análise do funcionamento dos protocolos e procedimentos relacionados que suportam este funcionamento. As idéias gerais foram mostradas nas seções anteriores, mas alguns conceitos serão ainda mais detalhados nesta seção, para que as funcionalidades apresentadas sejam melhor compreendidas.

Deve-se ressaltar novamente que a estrutura mostrada nesta seção é *CONCEITUAL*, tratando-se de um refinamento das estruturas propostas pela ISO em [ISO9545]. Estruturas apropriadas para implementações podem ser obtidas a partir deste modelo, mas provavelmente se diferirão em alguns aspectos, por exemplo, para se ganhar em eficiência no código gerado. Alguns comentários sobre a influência das idéias deste modelo na implementação do SISDI-OSI serão apresentados no capítulo 5.

2.4.1 As AEs

Quando se estabelece uma associação os objetos que estarão se comunicando são invocações de AEs e, não, AEs em si. No entanto o esquema de endereçamento usado no RM-OSI é baseado em SAPs, que são ligados a AEs. Especificam-se, portanto, endereços de AEs, mas localmente é preciso haver a determinação de uma invocação desta AE para efetivamente tratar a comunicação.

Como visto na seção 2.3.1, a AEI que solicita o estabelecimento da associação pode fornecer identificadores de invocação e títulos de AEs e APs para se especificar exatamente o objeto com que deseja trocar dados. Pode também só fornecer o endereço dos P-SAPs da entidade com que deseja comunicar. Há portanto a necessidade, no sistema receptor de uma solicitação de estabelecimento de associação, de, a partir de uma especificação de AE, determinar uma AEI adequada para o tratamento da associação, de acordo com as informações fornecidas pela AEI que requisitou a associação. Por exemplo, se foram fornecidos, além do endereço de apresentação, o título da AE e um identificador de invocação, a AEI remota fica completamente especificada e o sistema remoto tem que apenas direcionar o tratamento do estabelecimento da associação para esta AEI. Se, como outro exemplo, só foi fornecido o endereço de apresentação, o sistema remoto tem que localmente definir uma AEI. Pode, por exemplo, criar uma nova AEI ou criar um novo SAO em uma AEI já existente.

Pode-se portanto considerar que a cada AE estão associados o conjunto de suas AEIs e um módulo, chamado *SELECIONADOR DE AEI*, que seria o responsável pela execução destas operações, como mostra a figura 2.10. Este módulo precisa utilizar o ASE ACSE, pois as informações sobre AEs, AEIs, APs e APIs vêm em APDUs deste ASE. O módulo *SELECIONADOR DE AEI*

trata a primitiva A-ASSOCIATE.ind do ACSE e faz a determinação da AEI. Caso esta determinação não seja possível, este módulo não aceita o estabelecimento da invocação. Em caso contrário o controle do estabelecimento é passado à AEI determinada. O seu componente que prosseguirá o tratamento será mostrado na seção 2.4.4. Os dados que vierem através do SAP para esta associação serão encaminhados para esta AEI.

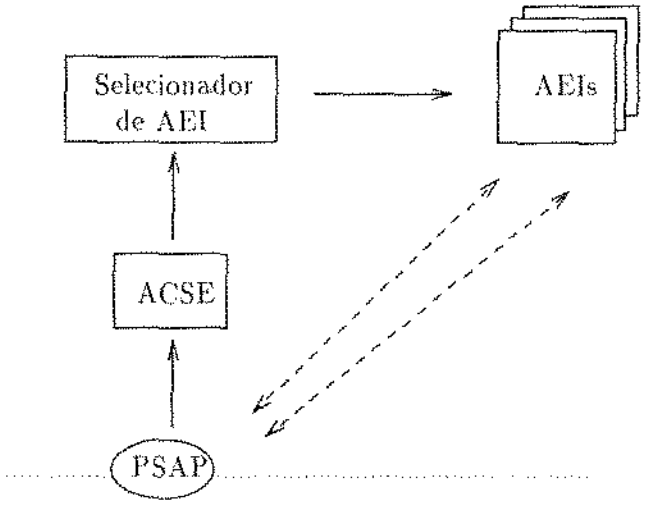

Figura 2.10: Modelo para AEs

O módulo *SELECIONADOR DE AEI* precisa possuir conhecimento sobre as AEIs, como, por exemplo, se há uma AEI com o identificador de invocação solicitado e qual o identificador da API ligada a elas. Além disto, conceitualmente pode-se considerar que pode haver várias instâncias do par formado por este módulo e pelo ASE ACSE associado, cada um deles tratando de uma solicitação de estabelecimento de associação específica.

2.4.2 AEIs

De acordo com o comentado na seção 2.1, segundo o modelo RM-OSI, uma AEI é composta de SAOs e MACF e modela funções de comunicação e informações de estado relacionadas ao processamento em suas associações. A princípio, portanto, foi concluído que uma AEI deveria possuir somente associações que tivessem relações entre si, relações estas que seriam dependências de estado na própria AEI. Associações independentes não possuem relacionamentos a modelar e, portanto, deveriam estar em AEIs diferentes. Desta maneira deveria haver somente um MACF por AEI, que coordenaria as dependências entre as associações.

No entanto foi mencionado que, para fins de eficiência, as AEIs poderiam ter um conjunto de associações que, embora não estejam ativas no momento, não são destruídas, uma vez que podem ser úteis no futuro e não se gastaria tempo em reestabelecê-las. Com isto pode-se ter dentro de uma AEI, usos de conjuntos distintos de associações, sendo que estes conjuntos não se relacionam. Um exemplo simples, onde estes conjuntos possuem uma só associação, seria o de uma AEI para transferência de arquivos possuir duas associações que não são liberadas após o uso. Estas associações podem ser usadas no futuro para a transferência de dois arquivos independentemente.

Estas observações levam à estrutura de AEI mostrada na figura 2.11. Nela estão representados os possíveis vários grupos de associações relacionadas, controladas por MACFs, mas cada conjunto sendo independente dos outros; um módulo chamado *INFO* que representa informações associadas à AEI, como o seu identificador de invocação e o identificador de invocação da API a que está associada; e um conjunto de associações controladas por um módulo chamado *GERENCIADOR DE ASSOCIAÇÕES LIVRES*. Este conjunto forma a coleção (*pool*) de associações da AEI mantidas conectadas para usos futuros. O módulo gerenciador deste *pool* é o encarregado de controlar estas associações, executando, por exemplo, atividades como atender solicitações de MACFs de pedidos de associações livres, selecionar uma associação que atenda às propriedades requisitadas nestas solicitações, que podem incluir quais unidades funcionais estão sendo usadas na associação e qual é o valor do parâmetro de qualidade de serviço da associação, etc.. Não se considerou aqui que esta funcionalidade constitui um MACF porque o MACF modela o controle de atividades relacionadas em mais de uma associação. Esta funcionalidade adicionada controla associações inativas, que, por isto, não podem ter atividades relacionadas com outras associações. Este módulo foi colocado na estrutura por exercer uma função que aparece nas descrições dos protocolos RDA [ISO9579-1] e TP [ISO10026-3] e que, provavelmente, aparecerá em descrições atualizadas de outros protocolos. Deve haver uma maneira de troca de informações entre os MACFs e o gerente do *pool* de associações livres.

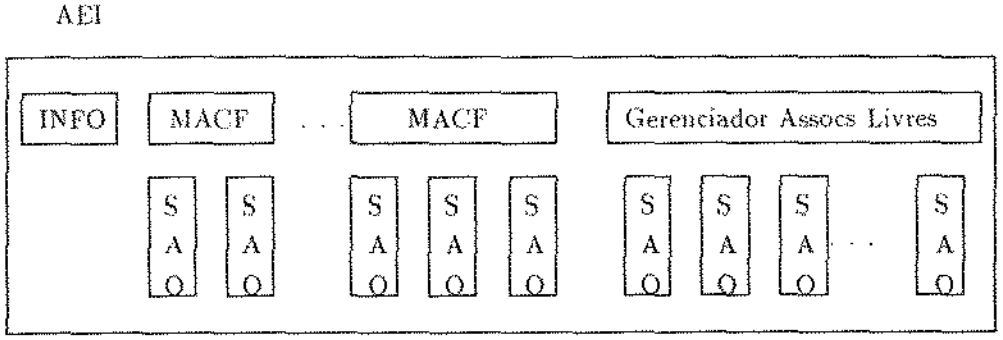

Figura 2.11: Modelo para AEIs

2.4.3 MACF

Como mencionado na seção 2.1, a ISO define MACF como sendo o conjunto das funções necessárias ao controle das atividades que ocorrem em mais de uma associação de uma AEI. Mais detalhadamente, no entanto, possui funções como:

1. serve como uma interface entre o protocolo e seu usuário, recebendo as primitivas que o usuário emite e executando as atividades necessárias, interagindo com os SAOs. Estas atividades podem estar relacionadas a uma única associação, ou com um sub-conjunto das associações. Ele também recebe eventos da máquina de protocolo (eventos vindos dos SAOs ou eventos internos), trata-os e envia primitivas para o usuário. Com estas funções o MACF pode gerar uma ou mais primitivas de serviços (enviadas aos SAOs) a partir de uma só primitiva do usuário e vice-versa;

2. é responsável por manter as regras de seqüenciamento de eventos em mais de uma associação;

3. mantém informação sobre o estado global de atividades executadas em múltiplas associações;

4. executa algumas atividades específicas do protocolo, por exemplo, colecionar *votos* no TP [ISO10026-3];

5. trata erros e toma as atitudes necessárias nas diversas associações.

A função central do MACF é, efetivamente, o controle do estado das diversas associações. As demais atividades que exerce são aquelas que tornam possível este controle ou que dependem de seus resultados. Em termos de funcionalidades, o MACF foi dividido nos componentes que aparecem na figura 2.12.

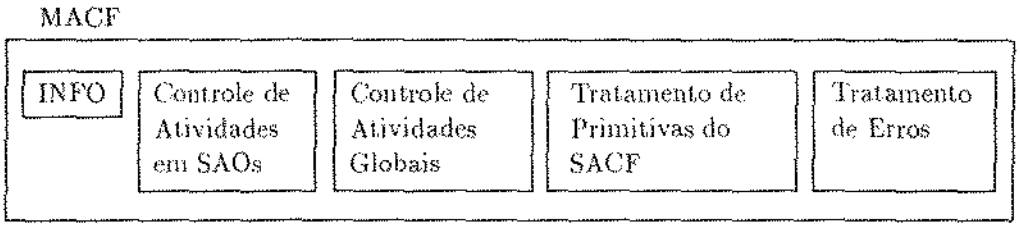

Figura 2.12: Modelo para MACFs

O módulo *INFO* é o que armazena informações necessárias ao ou decorrentes do controle das várias associações. Exemplos destas informações podem ser variáveis que indiquem o estado da atividade global executada nas associações e informações sobre primitivas anteriormente enviadas e seus parâmetros.

O módulo *CONTROLE DE ATIVIDADES EM SAOS* é o responsável por tratar ações que, embora se refiram a uma só associação, dependem do estado da atividade global, controlada pelo MACF. Por exemplo, após se decidir sobre o resultado final de uma transação no TP, diz-se que a máquina de estado do protocolo está no estado *Decidido* (do inglês *Decided*). Neste estado as entidades comunicantes não podem mais trocar dados de ASE usuário. Este módulo seria, por exemplo, o responsável por tratar eventos que surgissem a partir de violações desta regra, por parte da AEI com que se está comunicando. Conceitualmente haveria no MACF uma instância deste módulo para cada SAO.

O módulo *CONTROLE DE ATIVIDADES GLOBAIS* controla as ações que se referem à atividade global como um todo. No TP, por exemplo, uma ordem de efetivação da transação atômica é dada somente uma vez e o MACF se encarrega de distribuí-la nas diversas associações. Atividades deste tipo são exercidas neste módulo. Conceitualmente haveria somente uma instância deste módulo por MACF.

O módulo *TRATAMENTO DE PRIMITIVAS DO SACF* representa o tratamento de primitivas trocadas entre o SACF e o MACF para a coordenação de suas interações, por exemplo, servindo para o MACF comunicar ao SACF que o SAO no qual está não precisa mais ser controlado por ele, ou seja, não faz mais parte da atividade global. Este SAO pode, por exemplo, passar a fazer parte do *pool* de associações livres. Estas primitivas são de serviços fornecidos pelo SACF e se referem ao SAO como um todo.

O módulo *TRATAMENTO DE ERROS* cuida de eventos de erros que ocorrem em uma associação e que podem afetar as demais controladas pelo MACF. Por exemplo um erro em uma das associações no TP pode causar o *rollback* da transação e, com isto, fazer com que uma ordem de *rollback* deva ser transmitida em cada uma das demais associações.

Estes módulos se relacionam, fazendo com que atividades exercidas por um deles acarretem em atividades serem exercidas por outros.

2.4.4 O SACF

Conceitualmente a ISO define o SACF como sendo o elemento dentro do SAO que coordena a interação entre os diversos ASEs usados em uma associação e o uso dos recursos oferecidos pela camada de Apresentação (os recursos comentados na seção 2.3.4 são oferecidos à camada de Aplicação através da camada de Apresentação). Define ainda que as regras que regem esta coordenação estão definidas no contexto de aplicação que for usado.

Verificando mais detalhadamente o que o conceito de SACF representa e com base na descrição dos protocolos, principalmente o do protocolo TP [ISO10026-3], única especificação que apresenta explicitamente as atividades de um SACF, pode-se apresentar uma visão geral dos constituintes de um SAO da maneira mostrada na figura 2.13. Nesta figura pode-se observar os ASE associados ao SACF, da maneira apresentada em [ISO9545]. O SACF, por sua vez, está dividido em dois componentes: o módulo *ESTABELECEDOR DE ASSOCIAÇÃO* e o *PARTE ESPECÍFICA DE CONTEXTO DE APLICAÇÃO*. O primeiro deles representa as funções necessárias ao estabelecimento da associação, que, uma vez estabelecida, será controlada pelo segundo módulo.

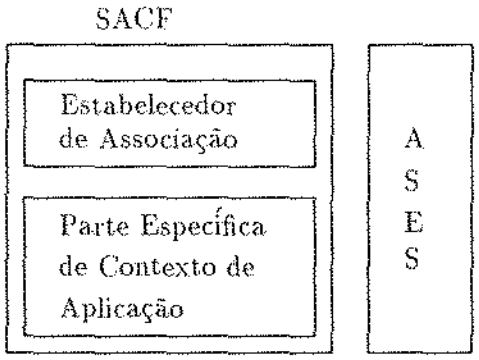

Figura 2.13: Modelo para SAOs

O módulo *ESTABELECEDOR DE ASSOCIAÇÃO* implementa regras de interação de ASEs e de uso de recursos da camada de Apresentação, mas que são utilizadas simplesmente no período de estabelecimento da associação. Na AEI que solicita o estabelecimento da associação, possui como funções, dentre outras, coletar APDUs dos ASEs, opcionalmente concatená-las, fornecer valores para alguns parâmetros de primitivas de serviços, obter informações de endereçamento e interagir com o ACSE através de suas primitivas de estabelecimento de associação. Nesta AEI pode requerer funções no MACF para obter informações de diretório, usando uma outra associação (ver seção 2.3.2). Na AEI receptora exerce atividades como verificar se os valores de alguns parâmetros

são aceitáveis, interagir com os ASEs para ver se as APDUs enviadas também são aceitáveis e enviar uma primitiva de serviço indicando a solicitação ao usuário do SAO (um AP ou um MACF). Controla também o envio da resposta do estabelecimento à AEI inicial.

Na fase de estabelecimento de associação se negocia o contexto de aplicação a ser usado. O módulo intitulado *PARTE ESPECÍFICA DE CONTEXTO DE APLICAÇÃO* passa então a controlar definitivamente as atividades no SAO e pode-se estruturá-lo, funcionalmente, de acordo com a figura 2.14. As funções específicas exercidas pelos módulos que aparecem nesta figura dependem do contexto de aplicação negociado, pois é ele quem descreve as regras utilizadas no SACF.

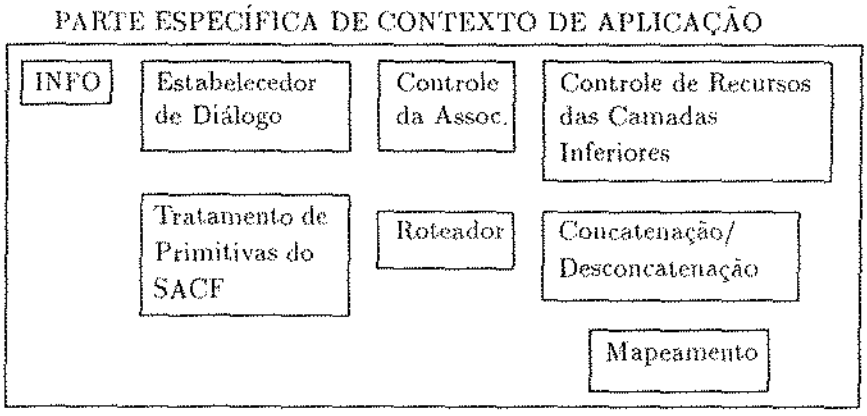

Figura 2.14: Modelo para a parte específica de contexto de aplicação de SACFs

De modo genérico, as funções dos módulos mostrados nesta figura são as seguintes:

INFORMAÇÕES este bloco contém as informações que são úteis para o SACF, como os estados resultantes das atividades na associação, informações sobre primitivas anteriormente recebidas e enviadas, unidades funcionais utilizadas em cada ASE, *tokens* da camada de Apresentação que o SACF possua no momento, entre outras. Estas informações podem influenciar nas ações realizadas pelo SACF na recepção de um determinado evento;

ESTABELECIMENTO DE DIÁLOGO este módulo é o responsável pelo estabelecimento de novos relacionamentos sobre a associação entre as invocações de AE. Exemplos deste tipo de relacionamentos são os diálogos do TP e RDA;

CONTROLE DA ASSOCIAÇÃO este módulo é o responsável por manter o estado da associação consistente, ou seja, ele analisa os eventos que ocorrem no SAO e determina seus efeitos no estado da associação. Este módulo examina as primitivas que passam pelo SAO. Algumas delas alteram o estado global da associação, enquanto outras simplesmente têm que ser repassadas aos ASEs ou ao MACF, dependendo do sentido da transmissão. As regras de seqüenciamento combinadas dos eventos de todos os ASEs e a sua sincronização também devem ser observadas por este bloco funcional;

TRATAMENTO DE PRIMITIVAS DO SACF pode ser necessário ao SACF fornecer primitivas para controlar a associação com um todo. Por exemplo, em [ISO10026-3] o SACF

fornece duas primitivas: uma é usada pelo MACF para indicar ao SACF quando a associação não mais será usada e a outra é usada pelo SACF para indicar ao MACF que o SAO não está mais sob seu controle. Este bloco funcional é responsável pelo tratamento destas primitivas;

CONTROLE DE RECURSOS DAS CAMADAS INFERIORES controla o uso de recursos das camadas inferiores, como *tokens*, números de pontos de sincronização, contextos de apresentação, dentre outros, comentados na seção 2.3.4. Monta, por exemplo, as primitivas de solicitação de *tokens*, estabelecendo valores para seus parâmetros. Estes valores são determinados em função do uso destes recursos por todos os ASEs do SAO, como, por exemplo, no caso de números de pontos de sincronização, que devem ser seqüenciais na associação, independentemente de qual ASE possa estar fazendo uso do ponto;

CONCATENAÇÃO e DESCONCATENAÇÃO o SAO é responsável por concatenar APDUs criadas nos ASEs para serem enviadas como dados de usuário de uma primitiva da Apresentação. Isto decrementa o número de primitivas da Apresentação enviadas, o que melhora o desempenho. As concatenações são feitas seguindo regras presentes no contexto de aplicação, e estas regras devem considerar a semântica das atividades nos SAOs e manter consistência de estado entre as diversas camadas do modelo. Durante a recepção de uma primitiva da Apresentação, o método inverso deve ser feito;

ROTEADOR este bloco funcional é responsável pelo direcionamento de primitivas geradas por ASEs, a partir de APDUs recebidas do sistema remoto, e de primitivas da camada de Apresentação, ao componente da AEI que deve tratá-las. Este componente pode ser algum ASE ou algum módulo do SACF, que pode, posteriormente, enviá-la ao MACF ou ao AP. Esta funcionalidade é necessária, pois as primitivas que têm como dados de usuário APDUs de outros ASEs têm que ser enviadas a eles. Primitivas sem dados de usuário podem, por exemplo, serem transmitidas ao módulo de *CONTROLE DA ASSOCIAÇÃO*, se afetarem o estado da associação, e depois enviadas ao MACF. Esta funcionalidade torna também as funções dos ASEs mais modulares em implementações, porque estes podem assim prestar os serviços independentemente de qual componente da AEI trate cada primitiva específica;

MAPEAMENTO este bloco tem a função de mapear APDUs (concatenadas ou não) no parâmetro de dados de usuário de primitivas da Apresentação e também preencher os valores de outros parâmetros destas primitivas. No contexto de aplicação há regras especificando que primitiva deve ser usada e como os parâmetros devem ser preenchidos..

O SACF, em resumo, controla os aspectos da associação que dependem exclusivamente das atividades realizadas em uma associação, estando ela associada a um MACF ou não.

Capítulo 3

Estrutura Geral de Implementação dos Protocolos do SISDI-OSI

O objetivo principal deste capítulo é apresentar a estruturação geral de implementação dos ASEs, SACFs e MACFs, ou seja, apenas dos protocolos de Aplicação que fizerem parte do SISDI-OSI, em termos de processos e mecanismos de comunicação utilizados. Com exceção da parte da seção 3.5.2 que se refere a estrutura de APDUs no *buffer*, todas as outras seções deste capítulo, no entanto, aplicam-se também às implementações dos protocolos de Apresentação e Sessão.

A estrutura do SISDI-OSI foi definida tendo-se em vista seus objetivos didáticos e de que deveria ser bastante flexível, podendo inclusive suportar mais de uma metodologia ou ferramenta de desenvolvimento de implementações. Neste sistema são levados mais em consideração aspectos de clareza e simplicidade do que questões de eficiência.

Na seção seguinte se comenta sobre a estrutura geral das implementações dos protocolos; a seção 3.2 trata da estruturação destes em termos de processos, enquanto as seções 3.3 e 3.4 mostram a estrutura geral de comunicação usada, a saber, através de filas de mensagens e uma área de memória compartilhada. A seção 3.5 apresenta mais detalhadamente como esta estrutura é utilizada para a invocação de primitivas de serviços e armazenamento de PDUs.

3.1 A Estrutura Geral dos Módulos

A figura 3.1 mostra a estrutura geral de implementação dos ASEs, MACFs, SACFs e protocolos de Apresentação e Sessão. Cada um destes componentes é implementado como um processo UNIX independente. Cada processo possui uma fila, para a recepção de todas as mensagens enviadas de outros processos, e acesso a uma área comum de memória. Esta área comum é usada para armazenar parâmetros de primitivas de serviços e de controle, campos de PDUs e qualquer outra informação que for compartilhada por todos os processos.

Este esquema é apenas uma orientação para as implementações do SISDI-OSI. Algumas alterações podem ser feitas, caso uma implementação específica necessite, embora se espere que este seja o esquema realmente utilizado na maioria delas. Um exemplo destas alterações seria um processo ter mais de uma fila.

As seções seguintes comentam mais sobre cada item desta figura.

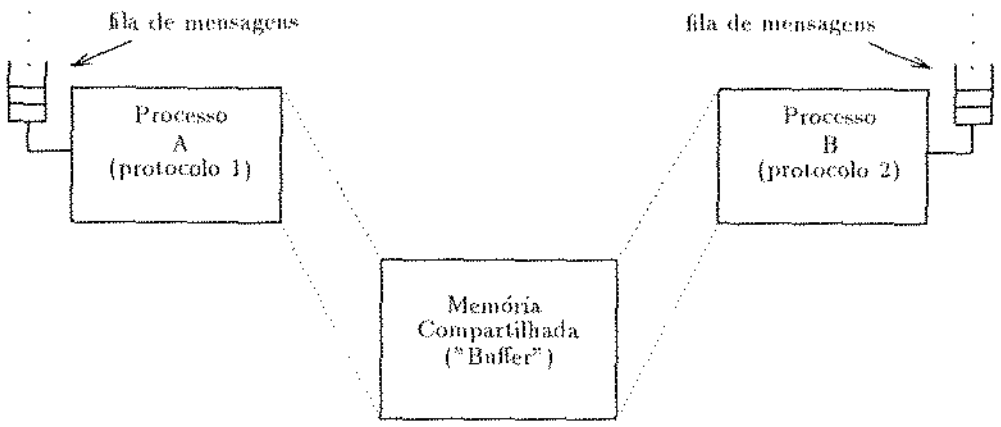

Figura 3.1: Esquema Geral de Implementação dos Protocolos

3.2 A Estrutura dos Protocolos em Processos

Uma estrutura em termos de processos para o SISDI-OSI deve atender aos seguintes objetivos:

1. as implementações dos protocolos devem ficar bastante localizadas, no sentido de que a lógica e as estruturas de dados utilizadas para a implementação de um dado protocolo devem ser facilmente diferenciadas das utilizadas para os demais. Isto é particularmente importante porque os protocolos não só são especificados separadamente, como também possuem complexidade suficiente que torna a tarefa de estudá-los isoladamente a forma mais prática ou mesmo viável de compreendê-los;

2. deve ser flexível o suficiente de modo a permitir que os implementadores experimentem ferramentas e métodos de implementação diferentes dos que os demais utilizem;

3. deve facilitar a substituição e inserção de implementações, que podem ser inclusive realizadas em outros ambientes;

4. deve facilitar a utilização de implementações do SISDI-OSI em outros sistemas.

Os pontos acima são decorrentes dos objetivos do SISDI-OSI. A flexibilidade em se permitir a remoção e inserção de implementações para e de outros ambientes é uma importante característica para que o sistema efetivamente seja uma plataforma para desenvolvimento e estudo de implementações que seguem os padrões do RM-OSI.

Considerando-se os princípios acima, quatro alternativas foram inicialmente consideradas para se estruturar o sistema:

1. cada protocolo ser implementado como um único processo, que simularia internamente as várias instâncias de uso deste protocolo, nas diversas associações;

2. cada instância de uso de um protocolo ser implementada como um processo independente;

3. cada instância de uso de um protocolo ser implementada como um processo, de maneira análoga ao item anterior, mas haver um número fixo destes processos; e

4. utilizar processos *leves* (*light-weight processes*). Processos leves são um mecanismo fornecido em alguns sistemas operacionais (em particular é fornecido na versão do sistema operacional utilizado na implementação) que permite a estruturação de um processo em várias *linhas de execução* (*threads*) independentes. Isto faz com que se possa criar um esquema de escalonamento interno a um processo, controlado pelo próprio usuário, de maneira que se possam diminuir os efeitos da troca de contextos de processos do sistema operacional, nos casos em que estes só utilizariam uma pequena parcela do tempo de CPU alocado a eles pelo escalonador do sistema.

Os itens 1 a 3 aparecem como sugestões em [Svo89]. Neste artigo outras alternativas são sugeridas, mas que se baseiam em ambientes de execução específicos. A alternativa 4 é apenas uma maneira particular de implementação de 1.

Dentre as alternativas, a opção escolhida foi a 1. Os itens 2 e 3, embora tenham a vantagem de tornar bastante independente as atividades executadas em cada instância, possuem as seguintes desvantagens:

1. utilizam mais recursos do sistema, não só porque resultam em mais processos serem executados na máquina, como por usarem recursos para a criação de mais filas, pois cada processo deveria ter a sua própria;

2. o item 3 implica em se ter funcionalidades adicionais para gerenciar o conjunto de processos que tratariam as instâncias de um determinado protocolo, para determinar, por exemplo, qual está livre para atender a uma nova instância de uso;

3. fazendo com que várias instâncias sejam controladas por um único processo faz com que se diminuam as trocas de contexto de processos no sistema, uma vez que, ao ser escalonado para ser executado, cada processo trataria todas as solicitações de serviços do protocolo, em várias instâncias;

4. o item 1 não cria dificuldades para se implementar independência entre as diversas instâncias de uso de um protocolo. Permite inclusive que para isto sejam utilizados recursos da própria linguagem que for usada para a implementação, caso esta forneça algum que suporte a definição destas atividades independentes. A linguagem ESTELLE, descrita no anexo A, usada para a implementação do CCR, fornece tais recursos. Na implementação do protocolo ACSE, no entanto, feita em C, que é uma linguagem que não fornece este tipo de facilidade, foram utilizadas tabelas contendo informações de estado de cada instância.

Quanto ao uso de processos *leves*, *threads* poderiam ser usados para implementar uma instância de um protocolo, internamente a um processo, ou para implementar um protocolo. Neste último caso, um processo conteria o código de mais de um protocolo, cada um deles sendo um *thread*. Mas a simulação interna de instâncias feita com o uso de tabelas, por exemplo, é mais simples de ser realizada.

Ao nível de processos, o uso de *threads* vai resultar em maior dependência entre as implementações dos protocolos que fizerem parte do processo. A implemementação deve ser feita de

modo que um protocolo não bloqueie a execução de outro, resultando na necessidade de se fazer um controle, por exemplo, na leitura de mensagens. O esquema de um protocolo por processo, com uma fila dedicada, é mais simples e permite facilmente a substituição e remoção (para outros ambientes) de implementações de um protocolo, por não haver tais dependências. Para ganho em eficiência, no entanto, o uso de processos *leves* parece ser útil.

3.3 A Estrutura de Comunicação

O paradigma de comunicação entre processos que é utilizado no SISDI-OSI é o de troca de mensagens, pois é um esquema que se deriva imediatamente do modelo de primitivas de serviços presente no modelo RM-OSI para comunicação entre as entidades. Além disto é um paradigma extremamente simples e é o que foi usado no SISDI-MAP, o que facilita a migração dos protocolos já implementados para esta nova versão do sistema.

O mecanismo utilizado para atender a este paradigma foi o de mensagens (*Messages*) do UNIX System V [Bac86, SUN90]. Algumas características deste mecanismo são:

1. é um mecanismo de filas, nas quais as mensagens ficam armazenadas. Vários processos podem ler e escrever em uma mesma fila;

2. as mensagens são estruturadas, ou seja há uma delimitação de início e fim de cada mensagem, ao contrário da filosofia de *stream* de dados, usada em outros mecanismos, onde as mensagens chegam ao sistema receptor como uma seqüência não estruturada de *bytes*;

3. permite que seja atribuído um tipo a cada mensagem e que a seleção de mensagens a serem lidas seja feita de acordo com este tipo;

4. permite que os processos "durmam" automaticamente, tanto no envio quanto na recepção de mensagens. Um processo que quer enviar uma mensagem "dorme" se não houver espaço disponível na fila. O sistema automaticamente o "acorda" quanto passar a haver. De maneira análoga, um processo que estiver lendo uma fila automaticamente "dorme" se não houver mensagens nela. O sistema o "acorda" quando passar a haver;

5. não possui restrições quanto a processos que podem escrever ou ler na fila (salvo um esquema de permissões análogo ao de arquivos), ou seja, não é necessária, por exemplo, uma relação de ancestralidade entre os processos que fizerem uso da fila;

6. é implementado na memória principal, não utilizando o sistema de arquivos.

Este mecanismo é bastante adequado aos requisitos de comunicação de uma implementação de um protocolo. O fato de fornecer um esquema de filas faz com que os serviços possam ser solicitados de maneira assíncrona. A possibilidade de se atribuir tipos a mensagens e lê-las segundo estes tipos permite que se implementem mecanismos de prioridade para primitivas, que são úteis, por exemplo, para a implementação dos serviços de dados com urgência (*expedited*) das camadas de Apresentação, Sessão e Transporte. O esquema no qual os processos são "postos para dormir" e são "acordados" evita o *polling* na fila, fazendo, portanto, um melhor uso de recursos do sistema e já implementando um mecanismo de controle de fluxo entre protocolos. A não exigência de relacionamentos como o de ancestralidade entre os processos que fazem uso da fila é útil já que os relacionamentos de

trocas de mensagens entre os processos da camada de Aplicação são complexos, não obedecendo a nenhum tipo de estrutura específico. O fato de ser implementado na memória, embora não seja de primeira importância para o SISDI-OSI, confere um melhor desempenho.

Outros esquemas de comunicação entre processos são fornecidos por sistemas compatíveis com UNIX, como, por exemplo, *pipes*, *named pipes* e *sockets* [Bac86]. Estes esquemas, no entanto, não oferecem todas as vantagens do esquema de *Messages*. Por exemplo, utilizam a filosofia de *stream* de dados, não fornecendo o esquema de seleção de mensagens por tipo, ou, no caso de *pipes*, exigem uma relação de ancestralidade entre os processos que fazem uso de uma fila.

O esquema geral propõe que cada processo tenha uma única fila, exclusiva, através da qual são recebidas todas as mensagens enviadas a ele. Este esquema tem vantagens sobre outros que utilizam mais de uma fila, pois otimiza recursos do sistema e permite que se explore o fato de os processos serem colocados para "dormir". Com mais de uma fila um esquema de *polling* teria que ser feito para se verificar se há alguma mensagem em alguma das filas. No entanto, como mencionado anteriormente, o esquema geral define apenas uma estrutura básica para as implementações, permitindo, caso seja julgado interessante pelo implementador, que mais de uma fila seja usada.

O esquema de troca de mensagens entre processos no SISDI-OSI utiliza as filas juntamente com um esquema de memória compartilhada. Como se verá na seção 3.5, a mensagem que é colocada na fila possui campos que são usados apenas para identificar e indicar onde está o seu conteúdo de informação, que fica alocado em um *buffer* implementado nesta área de memória compartilhada. O mecanismo de *Shared Memory*, também do UNIX System V, é utilizado para este fim. A estrutura das mensagens colocadas nas filas será mostrada na seção 3.5.1 adiante.

3.4 A Estrutura da Área Comum (*BUFFER*)

Todos os processos que implementam protocolos no sistema terão acesso a uma área de memória compartilhada que serve para armazenar informações utilizadas por vários processos. Na implementação atual esta área contém apenas um *buffer* para o armazenamento de primitivas e PDUs.

Para a criação desta área de memória foi utilizado o mecanismo de *Memória Compartilhada* (*Shared Memory*), do UNIX System V [Bac86, SUN90]. No SISDI-OSI, a área de memória é inicialmente criada com um tamanho suficiente para as necessidades do sistema e inicializada com os campos necessários para a implementação do *buffer*. Depois cada processo do sistema, ao se "inicializar", acopla-a a seu espaço de endereçamento.

O *buffer* para primitivas e PDUs é implementado como uma única área contínua de dados e possui todas as suas estruturas de dados de controle na própria área compartilhada. Foram implementadas para o sistema rotinas de acesso ao *buffer*, para serem utilizadas por todas as implementações. A tabela 3.1 apresenta quais são estas rotinas e uma breve descrição delas. Para que um processo tenha acesso a estas rotinas basta que seu código seja *link-editado* com os códigos delas.

Para se determinar um bom algoritmo de alocação e desalocação de áreas para um *buffer* qualquer, deve-se ter em mente as características da aplicação que estiver fazendo uso dele. Fatores que influenciam na escolha são a faixa de tamanho de pedidos de alocação, os padrões de alocações e desalocações, a distribuição de probabilidade com que cada tamanho de área de alocação é pedido, dentre outros. Não foi feita uma análise deste tipo para o SISDI-OSI e portanto foi implementado apenas um algoritmo genérico. Este algoritmo utiliza o de *liberação com* tags *nas bordas* (*Liberation*

Rotina	*Descrição*
buffer_inicializa	monta as estruturas de dados para implementação do *buffer* na área compartilhada e "inicializa" valores
buffer_conecta	conecta a área compartilhada no espaço de endereçamento do processo
buffer_aloca	aloca uma área no *buffer*
buffer_desaloca	desaloca uma área do *buffer*
buffer_diminui_area	desaloca um trecho de uma área do *buffer* anteriormente alocada
buffer_descr_para_apontr	converte um descritor em um apontador (ver capítulo 5)
buffer_apontr_para_descr	converte um apontador em um descritor
buffer_desaloca_descr	desaloca uma área do *buffer* referenciada por um descritor
buffer_aloca_descr	aloca uma área no *buffer* e retorna um descritor

Tabela 3.1: Rotinas para manipulação do *Buffer*

with boundary tags) para desalocação, apresentado em [Knu73], aliado a um algoritmo tradicional de alocação *first-fit*, com uma fila de áreas livres, não ordenada [Knu73], adaptado para atualizar os campos dos cabeçalhos necessários para o algoritmo de desalocação. O algoritmo resultante possui complexidade linear no número de áreas livres para a alocação e constante para desalocação.

Uma característica do padrão de uso do *buffer* pelos protocolos é que, uma vez alocada uma área no *buffer*, o uso desta área será feito por um único componente do sistema por vez. Isto faz com que não seja necessário um mecanismo de controle de concorrência para leituras e escritas no *buffer*, mas somente para as operações de alocação e desalocação, pois alteram dados de controle do *buffer* (número de *bytes* livres, dados de controle de cada área alocada, etc.). Este controle é feito com o uso de um *Semáforo*, implementado utilizando o mecanismo *Semaphores*, também do UNIX System V [Bac86, SUN90]. As rotinas para controle deste semáforo são chamadas internamente pelas rotinas de alocação e desalocação de áreas do *buffer* e pela rotina que diminui uma área alocada. A tabela 3.2 apresenta as rotinas implementadas.

O esquema de estruturação das áreas alocadas do *buffer* será mostrado na seção 3.5.2.

3.5 Utilização dos Mecanismos de Comunicação pelos Processos

Nesta seção será descrita a estruturação feita nas mensagens que são colocadas nas filas e nas áreas alocadas no *buffer*.

Rotina	*Descrição*
obtem_semaforo	obtém o identificador do semáforo binário para o controle de concorrência ao *buffer*
inicializa_semaforo	inicializa o semáforo binário com um valor fornecido como parâmetro
P	função para entrada na área crítica. Diminui o valor do semáforo de 1
V	função para a saída a área crítica. Aumenta o valor do semáforo de 1

Tabela 3.2: Rotinas para manipulação do semáforo para controle de concorrência ao *buffer*

3.5.1 Estrutura das Mensagens na Fila

A estrutura geral de uma mensagem enviada pelo mecanismo *Messages* do UNIX é a mostrada na figura 3.2.

Figura 3.2: Estrutura geral de mensagens do mecanismo *Messages*

O campo *tipo* é o utilizado para a atribuição de tipos às mensagens e pode ser usado, como já indicado anteriormente, para implementar um esquema de prioridade para as mensagens das filas. No entanto, para a camada de Aplicação, somente o campo de dados é utilizado (pelo menos por enquanto. Pode ser que características particulares do funcionamento de algum protocolo ou mecanismo desta camada necessite deste campo). Independentemente da camada, o campo de dados é estruturado da maneira mostrada na figura 3.3.

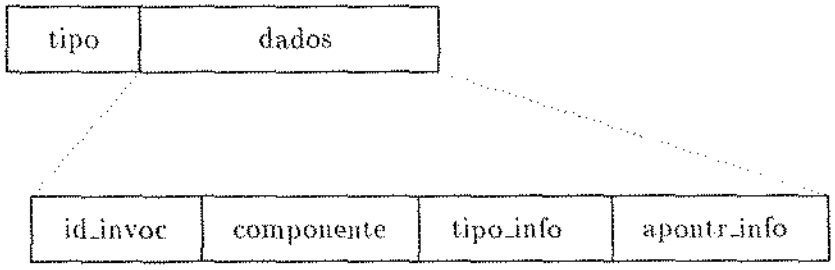

Figura 3.3: Formato das mensagens colocadas nas filas

Lembrando-se de que no sistema há um processo por protocolo, que simula internamente as suas várias instâncias de uso, e de que a mensagem que se coloca nas filas dos processos somente

serve para identificar a informação que se quer transmitir e indicar onde no *buffer* ela se encontra, os campos mostrados na figura 3.3 possuem as seguintes finalidades:

id_invoc é o identificador da invocação do protocolo à qual a mensagem que se está enviando se destina;

componente indica a qual módulo (TP-ASE, ACSE, CCR, RDA, SACF, APRESENTAÇÃO, etc.) do sistema a mensagem está relacionada. Se a mensagem é uma primitiva ou uma PDU de um protocolo X, este campo terá o valor X. Observe que este campo não indica, em si, nem o emissor, nem o receptor da mensagem. Por exemplo, quando o TP invoca um serviço do CCR, ele vai colocar uma mensagem na fila do CCR, que terá o seu campo *componente* com o valor *CCR*, pois se trata de uma primitiva deste protocolo. Da mesma maneira o CCR, ao enviar uma APDU sua para um SACF, colocará uma mensagem na fila do processo que implementa este SACF com o seu campo *componente* também com valor igual a *CCR*, pois se trata de uma APDU deste protocolo;

tipo_info identifica o tipo de informação da mensagem. Pode possuir somente um dos dois valores: *PRIMITIVA* ou *PDU*; e

apontr_info é um apontador para a área do *buffer* onde efetivamente se encontra o conteúdo da mensagem transferida. Nesta área poderão estar os valores dos parâmetros de uma primitiva ou dos campos de uma PDU, dependendo do valor do campo *tipo_info*.

A figura 3.4 ilustra o uso destes campos, com um exemplo no qual o serviço C-COMMIT é solicitado à invocação de número 4 do CCR. Uma primitiva C-COMMIT.req deve portanto ser enviada ao processo que implementa este protocolo. Como será visto posteriormente, quando uma primitiva é armazenada no *buffer*, o primeiro campo identifica o tipo desta primitiva. Esta identificação é feita utilizando-se uma constante inteira, que é dependente de cada implementação de protocolo no SISDI-OSI. A primitiva C-COMMIT.req é a de número 9 na implementação do CCR.

A escolha de quais informações seriam colocadas nas mensagens da fila e quais ficariam na área alocada do *buffer* foi feita de modo a evitar cópias de grandes quantidades de dados e que as informações mantidas no *buffer*, no caso de APDUs, ficassem já prontas para serem codificadas. Fazendo com que a mensagem na fila contenha um apontador para o local no *buffer* onde realmente a informação está permite que o processo gerador dos dados os escreva na área compartilhada e o(s) componente(s) do sistema que precisar(em) acessá-los basta que o façam através de apontadores. A informação mantida no *buffer* está descrita na seção seguinte.

De modo análogo ao caso do *buffer*, há implementadas no sistema rotinas para acesso às filas, que são *link-editadas* com o código das implementações dos protocolos. As rotinas implementadas aparecem na tabela 3.3.

3.5.2 A Estrutura das Áreas Alocadas do *Buffer* : Armazenamento de PDUs e Primitivas

As áreas alocadas no *buffer* servirão para armazenar parâmetros de primitivas e campos de PDUs. Esta seção trata de como estes dados são estruturados.

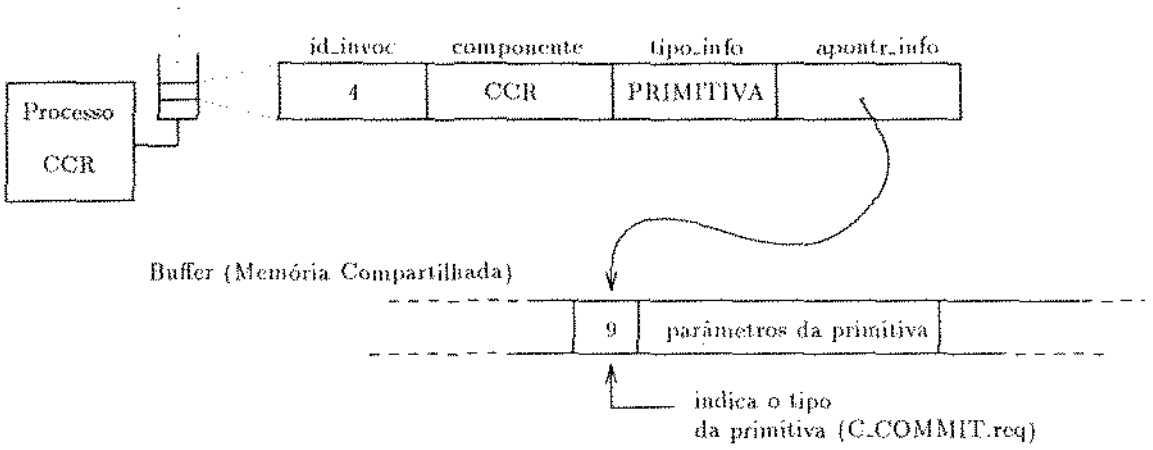

Figura 3.4: Exemplo do uso dos campos das mensagens colocadas nas filas

Rotina	*Descrição*
fila_obtem_id_fila_tp	obtém o identificador da fila associada ao processo que implementa o protocolo TP
fila_obtem_id_fila_sacf	obtém o identificador da fila associada ao processo que implementa o SACF (ver capítulo 5)
fila_obtem_id_fila_ccr	obtém o identificador da fila associada ao processo que implementa o protocolo CCR
fila_obtem_id_fila_acse	obtém o identificador da fila associada ao processo que implementa o protocolo ACSE
fila_obtem_id_fila_apresentacao	obtém o identificador da fila associada ao processo que implementa o protocolo orientado a conexão da camada de Apresentação
fila_envia_mensagem	rotina para envio de mensagem
fila_le_mensagem	rotina para leitura de mensagem de uma fila
fila_numero_mensagens	indica o número de mensagens que estão em uma fila
fila_ha_mensagens	indica se há ou não mensagem em uma fila

Tabela 3.3: Rotinas para manipulação de filas

As PDUs da camada de Apresentação e inferiores possuem uma forma de codificação para serem enviadas na rede determinada nos próprios protocolos, e esta codificação é utilizada em todas as conexões da camada. Para cada tipo de campo especifica-se uma forma de codificação específica, como, por exemplo, um campo inteiro ser codificado em apenas um *byte*, em complemento de 2, ou caracteres serem codificados em ASCII.

No caso da camada de Aplicação isto não acontece. Nesta camada há uma grande variedade de tipos de dados, incluindo tipos com estruturas bem mais complexas que a dos campos das APDUs das camadas inferiores, como, por exemplo, *strings* de vários conjuntos de caracteres. Isto faz com que a existência de várias maneiras de codificação seja necessária, cada uma adequada para um certo tipo de conjunto de dados. As formas iniciais de codificação a serem usadas em uma associação são negociadas no momento de seu estabelecimento. Durante o tempo de vida desta associação, estas formas podem-se alterar. A camada de Apresentação é a que tem como função fazer esta negociação e codificação em si dos dados.

Para se permitir no modelo RM-OSI estas várias maneiras de codificação, as PDUs dos protocolos da camada de Aplicação são descritas de uma maneira abstrata, utilizando-se uma linguagem própria, chamada ASN.1 (*Abstract Syntax Notation 1*) [ISO8824]. Esta linguagem permite a descrição dos tipos dos dados de maneira independente da representação que terão em máquinas específicas. Possui também construtores que permitem que a codificação de qualquer tipo de dado seja especificada de forma não ambígua. Linguagens de programação de propósito geral, como C, não possuem esta característica.

A descrição abstrata de um conjunto de PDUs de um protocolo constitui o que se chama uma *sintaxe abstrata*. A codificação que efetivamente os campos das PDUs têm em uma máquina particular constitui a *sintaxe local* destes dados. Ao se estabelecer uma associação as AEIs negociam uma forma de codificar estes dados para a transferência na rede, o que vai constituir a *sintaxe de transferência* dos dados. Esta sintaxe de transferência pode ser igual a uma das sintaxes locais ou não. Escolhendo-se uma sintaxe de transferência adequada e fazendo-se conversões, caso necessário, da sintaxe local para a de transferência e vice-versa em cada AEI, obtém-se uma codificação comum para a transmissão, o que permite que cada AEI "entenda" os dados enviados pela entidade remota. Um par sintaxe abstrata, sintaxe de transferência é chamado de *contexto de apresentação*.

No SISDI-OSI é utilizado um compilador, descrito em [Res92], que, a partir de uma sintaxe abstrata em ASN.1, ou seja, a partir da descrição de um conjunto de PDUs de um protocolo nesta linguagem, gera:

1. uma descrição análoga das PDUs em linguagem C, armazenando as informações necessárias à obtenção da não ambigüidade na codificação em tabelas de símbolos internas, e

2. rotinas de codificação e decodificação para cada PDU, de acordo com um conjunto de regras de codificação chamado BER (*Basic Encoding Rules*), também definido pela ISO [ISO8825]. As sintaxes das PDUs de uma sintaxe abstrata, após a codificação de seus campos segundo estas regras, constituem uma sintaxe de transferência. BER será a única forma de codificação suportada pelo SISDI-OSI.

A descrição das PDUs em C define como os seus campos devem estar estruturados na memória para poderem ser processados pelas rotinas de codificação. Para cada tipo básico de ASN.1 o compilador estabelece um mapeamento para estruturas na linguagem C, que pode ser direto, como dados que representam valores inteiros, ou não, como no caso de conjuntos (tipo *SEQUENCE OF* de

ASN.1) ou de registros variáveis (tipo *CHOICE*, de ASN.1). O tipo *CHOICE* corresponde ao tipo *union* de C, ou seja, um registro cujo valor pode ser um de um conjunto de tipos especificados. O mapeamento do tipo *CHOICE* em C, por exemplo, ilustrado na figura 3.5, corresponde aos tipos do *CHOICE* acrescidos de um campo onde se identifica qual dos tipos foi realmente o utilizado dentre as opções. A descrição das PDUs em C é a descrição de seus campos segundo estes mapeamentos. A partir desta descrição, as rotinas de codificação podem converter os dados para a sintaxe de transferência.

```
Em ASN.1 :
  tipo_exemplo ::= CHOICE
                    {
                      ident_1 Tipo_1,
                      ident_2 Tipo_2,
                            .
                            .
                            .
                      ident_n Tipo_n
                     }
Em C:
typedef struct tipo_exemplo
 {
   int tipo;

   union
    {
      Tipo_1     ident_1;
      Tipo_2     ident_2;
                       .
                       .
                       .
      Tipo_n     ident_n;
    } componentes;
 } tipo_exemplo;
```

Figura 3.5: Mapeamento do tipo *CHOICE* em C

De maneira análoga, no momento da recepção de uma PDU, as rotinas de decodificação transformam as PDUs da maneira como foram codificadas para transmissão (sintaxe de transferência) para a sintaxe local de acordo com estas estruturas em C.

A camada de Aplicação permite que APDUs sejam concatenadas e que apareçam deste modo ou separadamente como dados de usuário de outras APDUs, de primitivas dos ASEs e de primitivas da camada de Apresentação. A maneira utilizada para se permitir estas estruturações pode variar de um sistema para outro. No SISDI-OSI foi utilizada a própria estrutura de dados gerada pelo compilador, que atende bem ao funcionamento desta camada. Os recursos necessários para

realizar estas operações com APDUs ficaram embutidos nas próprias estruturas em C. Os dados da Aplicação poderiam ter sido estruturados da maneira mais apropriada para o funcionamento dos protocolos e então convertidos para a estrutura definida pelo compilador. Isto não foi necessário no SISDI-OSI. Para os protocolos implementados em C, basta então que se inclua, no programa que implementa o protocolo, o arquivo com as estruturas geradas nesta linguagem. Com isto se terá, automaticamente, as estruturas de dados para a declaração das PDUs e acesso a seus campos.

Embora, como dito no início desta seção, a codificação das PDUs da camada de Apresentação seja especificada no próprio protocolo, de modo análogo à das camadas inferiores, utilizando sempre um único conjunto de regras, o compilador ASN.1 também será usado para se codificar estas PDUs. Isto acontece porque o conjunto de regras de codificação utilizado nesta camada é o próprio BER. O documento que especifica o protocolo orientado a conexão da camada de Apresentação [CCITT226], que é utilizado no SISDI-OSI, apresenta a descrição em ASN.1 destas PDUs. Esta descrição pode então ser utilizada da mesma maneira que as de APDUs para gerarem estruturas de dados em C.

As estruturas das primitivas ficam também, em parte, definidas pelas estruturas das PDUs, já que os campos das PDUs são preenchidos, quase que em todos os casos, pelos valores de parâmetros correspondentes nas primitivas. Já que os formatos de estrutura dos campos de PDUs são determinados pelo compilador, é útil que se estruturem os parâmetros correspondentes das primitivas da mesma maneira, para evitar conversões.

Há, no entanto, parâmetros em primitivas das camadas de Aplicação e Apresentação que estão relacionados a PDUs da camada de Sessão, que não são processadas pelo compilador ASN.1. Com isto não se geram estruturas em C automaticamente para estes parâmetros. Neste caso o implementador do protocolo escolhe uma maneira apropriada de estruturá-los.

Emissão de Primitivas

Quando um protocolo quer invocar uma primitiva de serviço de um outro protocolo, os seguintes passos devem ser seguidos:

1. aloca uma área do *buffer*. Esta área deve conter um campo inteiro indicando qual primitiva se está enviando, seguido dos parâmetros da primitiva;

2. coloca na fila do processo do protocolo cuja primitiva se está invocando uma mensagem com a estrutura mostrada na figura 3.3. O campo *tipo_info* terá o valor *PRIMITIVA*. O campo *apontador* aponta para a área no *buffer* onde os parâmetros da primitiva estão armazenados. A estrutura destes parâmetros é gerada de acordo com o comentado na seção anterior. Os campos *id_invoc* e *componente* são preenchidos como comentado na seção 3.5.1.

Pode haver parâmetros das primitivas que sejam apontadores, por exemplo, os parâmetros de *lista de definição de contextos de apresentação* que aparece em primitivas de serviços da camada de Apresentação [CCITT216]. Este parâmetro representa uma lista em que cada componente é um registro contendo a identificação de um contexto de apresentação, o nome de uma sintaxe abstrata e um apontador para o próximo elemento da lista. Na área alocada no item 1 acima haverá o primeiro elemento da lista, no local do parâmetro da primitiva. Este terá um apontador para o segundo elemento da lista, que está alocado em outra região do *buffer*. Esta outra área, por sua vez, também conterá um apontador que poderá apontar para mais outra região do *buffer*, representando o terceiro elemento da lista, e assim sucessivamente. Cuidado, portanto, deve ser tomado na desalocação das primitivas. Esta desalocação é feita pelo protocolo que as recebe.

Armazenamento de uma APDU no *Buffer*

Nesta seção será comentado como é o tratamento de uma PDU da camada de Aplicação. O tratamento das PDUs das outras camadas não faz parte deste trabalho.

As APDUs de um ASE são enviadas para o sistema remoto ou ocupando um campo de uma APDU de um outro ASE ou no parâmetro de dados de usuário de uma PDU da camada de Apresentação. No primeiro caso a APDU é transferida ao ASE através de um parâmetro de uma das primitivas de serviço deste ASE. No segundo, as APDUs são transferidas ao SACF para concatená-las, se for o caso, que as mapeia no parâmetro de dados de usuário de primitivas da camada de Apresentação (pode ser que na implementação de alguns contextos de aplicação os próprios ASEs façam este mapeamento). Na AEI receptora, ocorrem procedimentos inversos.

Como comentado anteriormente, as estruturas das APDUs seguem o mapeamento para a linguagem C, feito pelo compilador, a partir de suas definições de tipos em ASN.1. As APDUs são tratadas no SISDI-OSI simplesmente de modo que estas estruturas de dados possam ser montadas, levando-se em conta as atividades realizadas por ASEs e SACFs. Para que uma APDU de um ASE possa ser o valor de um parâmetro de uma primitiva de um outro ASE, a sua estrutura deverá seguir o tipo do campo correspondente nas APDUs deste ASE, já que parâmetros de primitivas e campos de PDUs que se mapeiam têm o mesmo tipo. No entanto, quando uma PDU for ser enviada ao SACF, ela deve seguir uma estrutura de acordo com a definição do tipo dos parâmetros de dados do usuário das primitivas da camada de Apresentação. Da mesma forma, estes parâmetros terão o mesmo tipo que os campos análogos das PDUs desta camada. A seguir as estruturas para cada um destes casos são detalhadas.

Como exemplo serão considerados os protocolos TP e CCR. Estes protocolos possuem campos de dados de usuário nas APDUs (o TP só tem este campo em algumas APDUs), que são definidos como sendo do tipo *SEQUENCE OF EXTERNAL*, de ASN.1. O tipo *SEQUENCE OF tipoX* significa que se trata de uma fila, sem limite definido, em que todos os componentes são do tipo *tipoX*. A estrutura em C gerada pelo compilador para o tipo *SEQUENCE OF EXTERNAL* aparece na figura 3.6. O campo *ptr* aponta para o elemento do tipo *EXTERNAL* e o campo *next* aponta para o próximo elemento da fila. O tipo *EXTERNAL* possui, como valor, qualquer um de um tipo não especificado, que pode, inclusive, não ser descrito por ASN.1. A semântica do tipo *EXTERNAL* corresponde exatamente à definição de um campo de dados de usuário de uma PDU de um protocolo (ou do parâmetro correspondente nas primitivas), ou seja, o campo pode possuir qualquer dado de seu usuário. O fato de haver uma fila de elementos do tipo *EXTERNAL* quer dizer que o RM-OSI permite que possa haver concatenação de APDUs como dados de usuário de primitivas de serviços de ASEs (e, portanto, de APDUs).

O tipo *EXTERNAL* é definido pelo compilador como um registro, como mostrado na figura 3.7. O campo *direct_reference* serve para identificar uma sintaxe abstrata; o campo *indirect_reference* serve para se especificar um identificador de um contexto de apresentação; o campo *data_valuedescriptor* serve para descrever o dado ao qual o tipo *EXTERNAL* se refere; os campos do tipo inteiro, com exceção do campo *indirect_reference*, servem apenas para indicar se o campo que os seguem está sendo utilizado ou não (assumem valor 1 se sim; 0, se não).

Pelo fato de a única forma de codificação suportada no SISDI-OSI ser através da utilização do conjunto de regras BER, não há necessidade de se indicar contextos de apresentação para uma dada sintaxe abstrata. Por esta razão o campo *indirect_reference* não será utilizado. O campo *direct_reference* terá como valor a identificação da sintaxe abstrata que contém a definição da PDU

```
Em ASN.1 :
  tipo_exemplo ::= SEQUENCE OF EXTERNAL;

Em C :
  typedef struct tipo_exemplo
         {
           EXTERNAL *ptr;
           struct tipo_exemplo *next;
         } tipo_exemplo;
```

Figura 3.6: Mapeamento do tipo *SEQUENCE OF EXTERNAL* em C

```
typedef struct EXTERNAL
        {
          int direct_referenceop;
          objectidentifier direct_reference;
          int indirect_referenceop;
          int indirect_reference;
          int data_valuedescriptorop;
          objectdescriptor data_valuedescriptor;

          struct
           {
             int tipo;

             union
              {
                ANY single_ASN1type;
                octetstring octet_aligned;
                bitstring arbitrary;
              } componentes;
           } encoding;
        } EXTERNAL;
```

Figura 3.7: Definição do tipo *EXTERNAL* pelo compilador ASN.1

a que o tipo *EXTERNAL* se refere. O campo *data_valuedescriptor* é opcional e não será usado no SISDI-OSI. Regras mais detalhadas sobre como os campos deste registro devem ser preenchidos são encontradas em [ISO8824].

O campo *encoding* contém o dado em si, que vai constituir uma PDU. Este campo é dividido em dois: *tipo* e *componentes*. *componentes* é um registro variável, com três componentes : *single_ASN1type*, *octet_aligned* e *arbitrary*. O campo *tipo* serve para identificar qual tipo se está utilizando. Esta escolha depende de que tipo de codificação está-se fazendo para a transmissão do dado. O componente *arbitrary* é usado quando a codificação do dado não gerar um número inteiro de octetos. Se a codificação gerar um número inteiro de octetos, tanto a opção *octet-aligned* quanto a anterior podem ser usadas. Quando o valor do dado for de um único tipo especificado em ASN.1 e se este dado for codificado segundo as mesmas regras que o tipo *EXTERNAL* como um todo, além das opções anteriores, a opção *single_ASN1type* também pode ser usada. Em cada caso trata-se de uma escolha local de implementação escolher uma ou outra opção dentre as possíveis [ISO8824]. Como no SISDI-OSI o tipo *EXTERNAL* é utilizado para indicar uma única APDU que pode ser definida usando ASN.1 e todos os dados serão codificados segundo BER, a opção *single_ASN1type* pode ser usada e é a que sempre será.

ANY representa um tipo qualquer ou não especificado ou especificado em alguma sintaxe abstrata utilizando-se ASN.1 [ISO8824]. O compilador ASN.1 representa o tipo *ANY* da maneira mostrada na figura 3.8. O campo *nometipo* é usado para indicar o tipo do dado do usuário. Para este caso em que o campo do tipo *ANY* é um dos campos do tipo *EXTERNAL*, o tipo de dado de usuário deve ser um entre os especificados na sintaxe abstrata identificada no campo *direct_reference*. O campo *nome_tipo* é preenchido com o nome do tipo nesta sintaxe abstrata, na forma de um *string* de caracteres (tipo *valor*). O campo *ptrany* aponta para o dado em si.

```
typedef char valor[80];

typedef struct ANY
        {
          valor nometipo;
          void *ptrany;
        } ANY;
```

Figura 3.8: Definição do tipo *ANY* pelo compilador ASN.1

Uma definição ASN.1 de uma sintaxe abstrata é feita de uma maneira *top-down*, ou seja, sempre há um tipo mais genérico formado, em geral, de outros tipos. Estes outros tipos, por sua vez, podem ser formados a partir de ainda outros mais simples. A definição segue desta maneira até se atingir os tipos básicos da linguagem. Na definição das sintaxes abstratas dos protocolos, o tipo mais genérico é o que define as PDUs deste protocolo, na forma de uma estrutura do tipo *CHOICE*, comentada anteriormente. Este tipo para a sintaxe abstrata do CCR, por exemplo, e sua estrutura análoga em C aparecem, respectivamente, nas figuras 3.9a e 3.9b. A diferença entre esta figura e a 3.5, é que ASN.1 permite que em um *CHOICE* apenas os tipos dos campos sejam explicitados, como ocorre na figura 3.9 e não na 3.5. Neste caso o compilador insere nomes de campos na estrutura em C, como aparece na figura 3.9b (*CCR_APDUSeur1*, *CCR_APDUSeur2*, etc).

```
CCR_APDUS ::= CHOICE
              {
                C-BEGIN-RI,
                C-BEGIN-RC,
                C-PREPARE-RI,
                C-READY-RI,
                C-ROLLBACK-RI,
                C-ROLLBACK-RC,
                C-COMMIT-RI,
                C-COMMIT-RC,
                C-RECOVER-RI,
                C-RECOVER-RC
              }
```

(a)

```
typedef struct CCR_APDUS
 {
   int tipo;

   union
    {
      C_BEGINRI      CCR_APDUSvar1;
      C_BEGINRC      CCR_APDUSvar2;
      C_PREPARERI    CCR_APDUSvar3;
      C_READYRI      CCR_APDUSvar4;
      C_ROLLBACKRI   CCR_APDUSvar5;
      C_ROLLBACKRC   CCR_APDUSvar6;
      C_COMMITRI     CCR_APDUSvar7;
      C_COMMITRC     CCR_APDUSvar8;
      C_RECOVERRI    CCR_APDUSvar9;
      C_RECOVERRC    CCR_APDUSvar10;
    } componentes;
 } CCR_APDUS;
```

(b)

Figura 3.9: Mapeamento do tipo das APDUs do CCR em C

Os campos do tipo *EXTERNAL* serão preenchidos justamente para identificar este tipo mais genérico das sintaxes abstratas. O campo *ptrany* do tipo *ANY* (figura 3.8) apontará para uma região do *buffer* que conterá um valor deste tipo. A figura 3.10 mostra um exemplo para o caso do CCR.

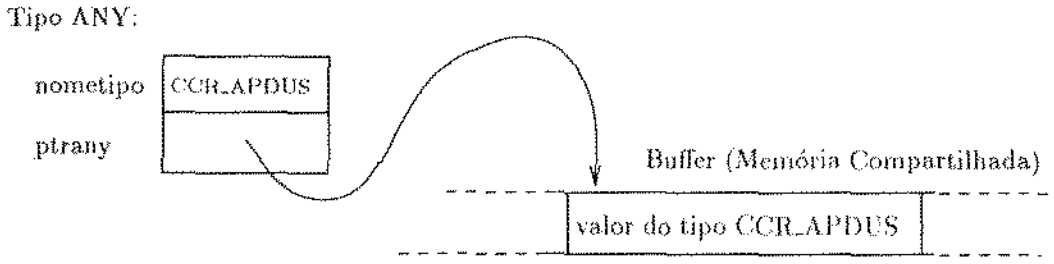

Figura 3.10: Exemplo do uso dos campos do tipo *ANY*

Em resumo, quando uma única APDU vai ocupar o campo de dados do usuário de uma primitiva de serviço de um dos ASEs TP ou CCR, o protocolo que a gerou armazena os campos da APDU e os parâmetros da primitiva em duas áreas distintas do *buffer*. O parâmetro de dados do usuário desta primitiva terá o tipo *SEQUENCE OF EXTERNAL*, que consiste de uma lista de elementos do tipo *EXTERNAL*. Um único elemento desta lista será usado, sendo preenchido da maneira descrita nos parágrafos anteriores. A figura 3.11 ilustra a situação resultante, em um exemplo em que a APDU TP-RECOVER-RI, do protocolo TP, é enviada como dado de usuário da primitiva C-RECOVER.req, do CCR, na invocação de número 3 deste protocolo. O número desta primitiva é 17 na implementação do CCR no SISDI-OSI. A seqüência de números *2-10-2-1*, que aparece no campo *direct-reference*, o valor *TP_APDUS*, que aparece no campo *encoding*, e o número 11 que aparece no campo *tipo* no retângulo mais abaixo da figura indicam, respectivamente, a sintaxe abstrata do TP, o tipo mais genérico que aparece nesta sintaxe abstrata e a APDU TP-BEGIN-DIALOGUE-RI no SISDI-OSI (o número 11 foi usado hipoteticamente, pois a implementação do TP ainda não está feita). O valor 1 do campo *tipo* do campo *encoding* especifica o uso do tipo *single_ASN1type*.

Quando uma concatenação de APDUs for feita para ser enviada como dados de usuário de uma primitiva de um dos ASEs TP ou CCR, os apontadores da estrutura mostrada na figura 3.6 são utilizados. Como comentado anteriormente, esta estrutura é gerada a partir da definição dos campos de dados de usuário das APDUs destes ASEs.

Para a camada de Apresentação a maneira de identificar o dado de usuário (uma ou mais APDUs) é um pouco diferente. A figura 3.12 apresenta a definição do tipo *User_data*, que é o tipo do campo de dados de usuário das PDUs (ou primitivas) desta camada, e alguns tipos usados nesta definição. Este tipo possui dois campos: o campo *tipo* e o campo *componentes*. O primeiro serve para indicar qual tipo será usado no campo *componentes*. O campo *componentes* é um registro variável que permite dois tipos, o *Simply_encodeddata* e o *Fully_encodeddata*. O primeiro tipo é usado quando só houver um contexto de apresentação definido para a associação e não se estiver usando a unidade funcional da camada de Apresentação que permite adicionar novos contextos, ou quando se estiver usando um contexto de apresentação *default* [CCITT216]. No SISDI-OSI só se usará o segundo tipo, pois haverá sempre no mínimo dois contextos sendo utilizados, um para a

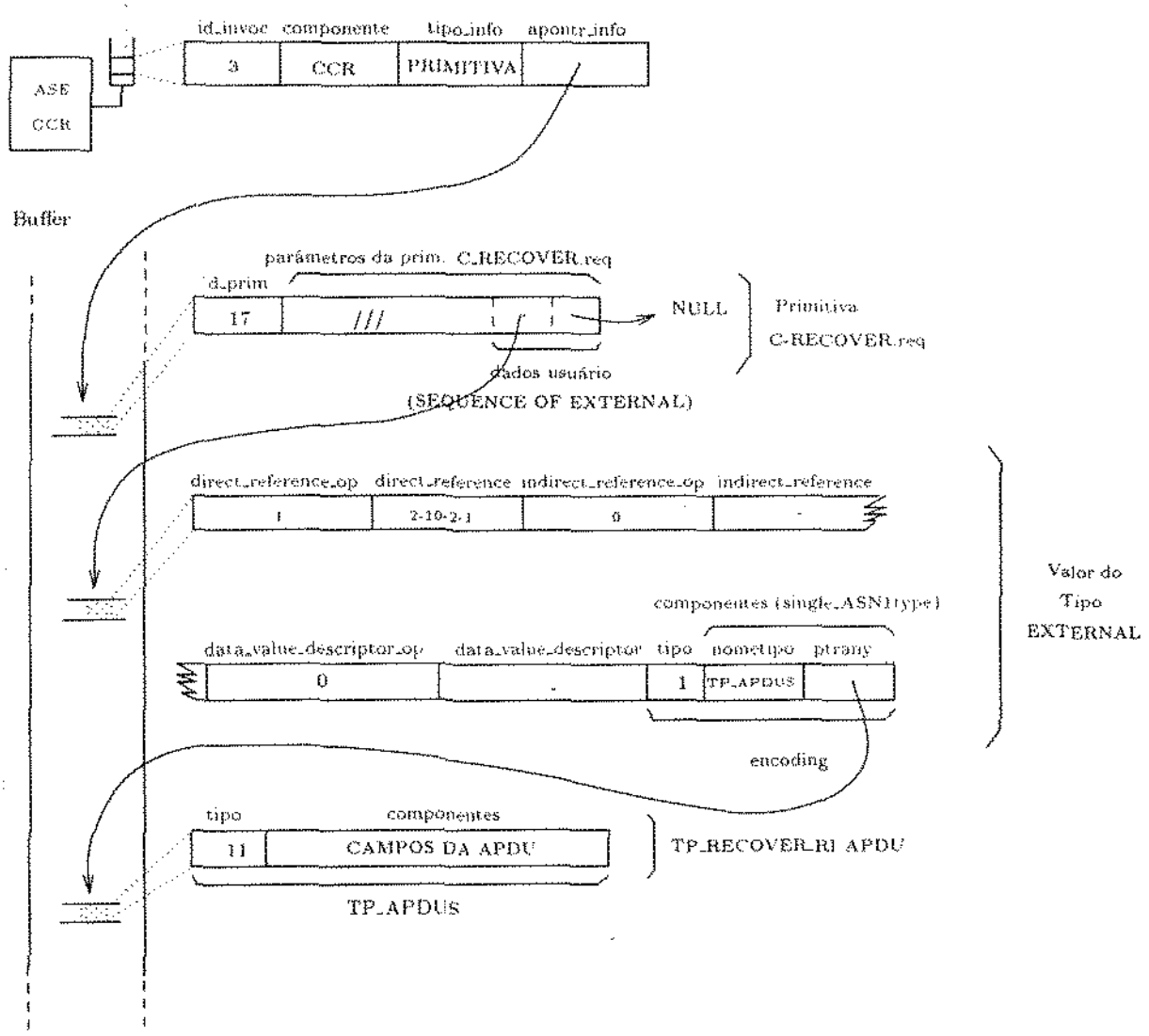

Figura 3.11: Exemplo de armazenamento de uma APDU no *buffer* para ser enviada como dado de usuário de uma primitiva de um ASE

sintaxe abstrata do ACSE e outro para a sintaxe abstrata do ASE que fornecer o serviço principal da associação, como, por exemplo, o RDA.

O tipo *Fully_encodeddata* é uma lista de valores, todos eles do tipo *PDV_list*. Cada elemento desta lista representa, no SISDI-OSI, uma APDU. A lista completa representa a concatenação destas APDUs.

No tipo *PDV_list*, o campo *PDV_listvar1op* é utilizado para indicar se o campo seguinte, que é opcional, está sendo usado ou não. O campo *PDV_listvar1* indica o nome da sintaxe de transferência a ser usada para codificar a APDU a ser transmitida. Como no SISDI-OSI só se usará o conjunto de regras de codificação BER, este campo não precisa ser preenchido. O valor do campo *PDV_listvar1op* portanto deve ser preenchido com o valor 0. O campo *presentation_datavalues* tem o mesmo tipo do campo *encoding*, utilizado no tipo *EXTERNAL*. No SISDI-OSI o campo da *union* contida em *presentation_datavalues* que é utilizado é o *single_ASN1type*. A razão disto é o fato de no SISDI-OSI o valor do campo *presentation_datavalues* conter sempre uma única APDU, especificada em ASN.1 e codificada segundo BER [CCITT226]. As condições nas quais cada um dos campos são usados são encontradas em [CCITT226]. De maneira análoga ao caso do tipo *EXTERNAL*, o campo *ANY* constitui-se de um apontador para a APDU e de um campo indicando o tipo desta APDU, que é um dos definidos na sintaxe abstrata do contexto de apresentação indicado no campo *PDV_listvar2*.

Os campos do tipo *PDV_list* são preenchidos também para identificar o tipo mais genérico das sintaxes abstratas, da mesma maneira que para o caso do tipo *EXTERNAL*. O campo *single_ASN1type* é preenchido, com isto, também do mesmo modo que no caso anterior.

Quando um ASE, portanto, for enviar uma APDU para o SACF, que fará o seu mapeamento no campo de dados de usuário de uma primitiva da camada de Apresentação, concatenada com outras APDUs ou não, basta que ele armazene esta APDU em uma área do *buffer* e envie uma mensagem ao SACF com um apontador para esta área. A figura 3.13 mostra, como exemplo, o envio de uma APDU C-RECOVER-RI, do CCR, ao SACF, na invocação de número 3. Observe nesta figura que há um apontador para a APDU TP-RECOVER-RI, que pode vir como dado de usuário da primitiva C-RECOVER.req, que gera esta APDU do CCR. A estrutura desta APDU do TP no *buffer* aparece na figura 3.11.

Ao receber uma APDU, o SACF irá então preencher os campos do tipo *PDV_list* de acordo com esta APDU. Estes campos, como dito anteriormente, fazem parte do campo do tipo *User_data* da primitiva na qual a APDU for enviada. Deve-se lembrar que o apontador do tipo *ANY* dos tipos *EXTERNAL* e *PDV_list* sempre aponta para o tipo mais genérico da sintaxe abstrata do ASE, que é um registro variável contendo todas as APDUs desta sintaxe abstrata. O campo chamado *tipo* que aparece na estrutura de dados em C equivalente a este registro serve para indicar o tipo da APDU.

A figura 3.14 ilustra a estrutura resultante destas atividades no SACF, mostrando como exemplo o que este faria ao receber a APDU da figura 3.13. A APDU C-RECOVER-RI é mapeada na primitiva P-TYPED-DATA.req da Apresentação (veja capítulo 5), identificada no SISDI-OSI pelo número 11, que aparece no primeiro campo *tipo* do local no *buffer* onde a primitiva está alocada. O campo *tipo* do campo *User_data* nesta mesma área do *buffer* indica que se está usando o campo *User_datavar2*, do tipo *Fully_encodeddata*, do campo *componentes* (um registro variável). Na área do *buffer* seguinte na figura o campo *PDV_listvar1op* tem valor 0, pois o campo *PDV_listvar1* não é usado, como indicado anteriormente. O campo *PDV_listvar2* possui valor 3, que é o valor que identifica o contexto de apresentação para a sintaxe abstrata do CCR no SISDI-OSI. O campo

```
typedef octetstring Simply_encodeddata;

typedef struct
 {
           int PDV_listvar1op;
           Transfer_syntaxname PDV_listvar1;
           Presentation_contextidentifier PDV_listvar2;
           struct
            {
              int tipo;
              union
               {
                 ANY single_ASN1type;
                 octetstring octet_aligned;
                 bitstring arbitrary;
               } componentes;
            } presentation_datavalues;
         } PDV_list;

typedef struct Fully_encodeddata
         {
           PDV_list *ptr;
           struct Fully_encodeddata *next;
         } Fully_encodeddata;

typedef struct User_data
         {
           int tipo;
           union
            {
              Simply_encodeddata User_datavar1;
              Fully_encodeddata User_datavar2;
            } componentes;
         } User_data;
```

Figura 3.12: Definição do tipo *User_data* em C

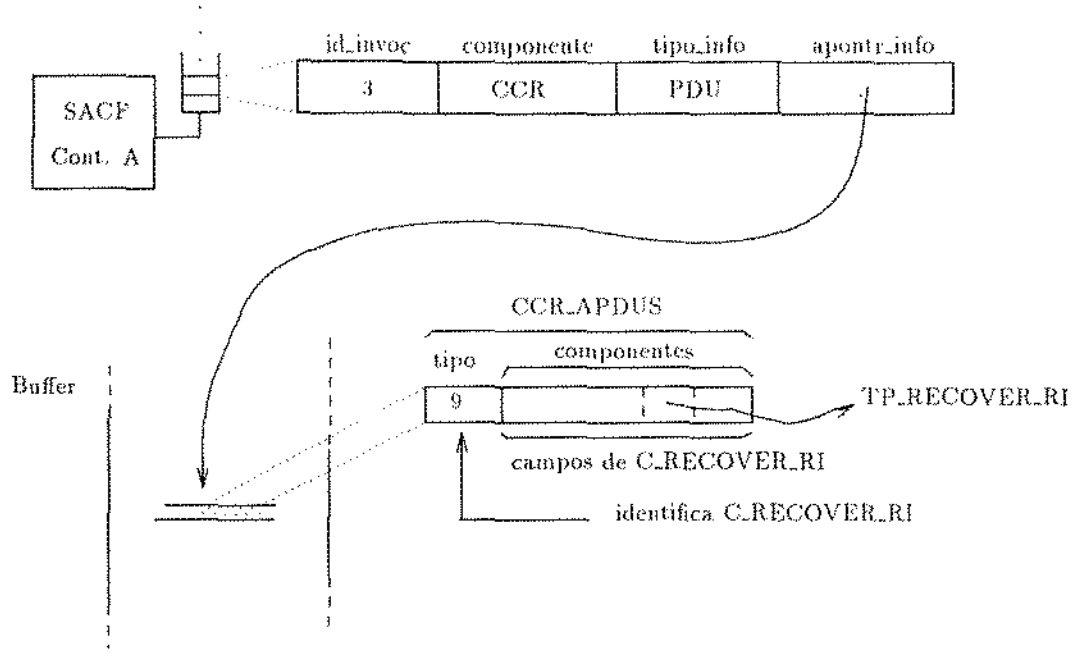

Figura 3.13: Exemplo de armazenamento de uma APDU no *buffer* para ser enviada ao SACF

tipo seguinte possui valor 1 para indicar que se está usando o campo *single_ASN1type* do campo *componentes* do campo *presentation_datavalues* (figura 3.12) . O valor *CCR_APDUS* que vem em seguida identifica que tipo de APDUs o apontador *ptrany* está apontando. A área do *buffer* mostrada mais abaixo na figura representa onde a APDU C-RECOVER-RI está armazenada e é a mesma que aparece na figura 3.13.

Os exemplos das figuras 3.11, 3.13 e 3.14 constituem a seqüência de construção das estruturas de dados para serem codificadas pelo compilador ASN.1 que ocorre desde a invocação da primitiva C-RECOVER.req com a APDU TP-RECOVER-RI como dado de usuário, até o mapeamento da APDU gerada em uma primitiva da Apresentação.

Para outros ASEs o procedimento á análogo, considerando-se o tipo do campo no qual as APDUs serão mapeadas.

As concatenações realizadas pelo SACF, como indicado anteriormente, são feitas utilizando o tipo *Fully_encodeddata* (figura 3.12). Pode ser o caso, no entanto, de que o ASE requisite que mais de uma APDU seja concatenada em uma mesma primitiva da camada de Apresentação. Se fossem passadas separadamente ao SACF, poderia ocorrer que este as concatenasse com outras APDUs, mas esta concatenação resultasse em as APDUs do ASE serem transferidas em primitivas diferentes. Para garantir isto elas são passadas juntas ao SACF. A mensagem que o ASE passa a ele contém um apontador para cada uma das APDUs, da maneira mostrada na figura 3.15. Nesta figura está um exemplo em que o CCR indica ao SACF a concatenação de duas APDUs, considerando que esta indicação está acontecendo na invocação de número 5 do SACF. Observe que foi criada uma primitiva especial para esta indicação de concatenação, que tem identificador número 26 para o CCR. As APDUs apontadas terão uma estrutura no *buffer* de modo análogo à da figura 3.13. Ao

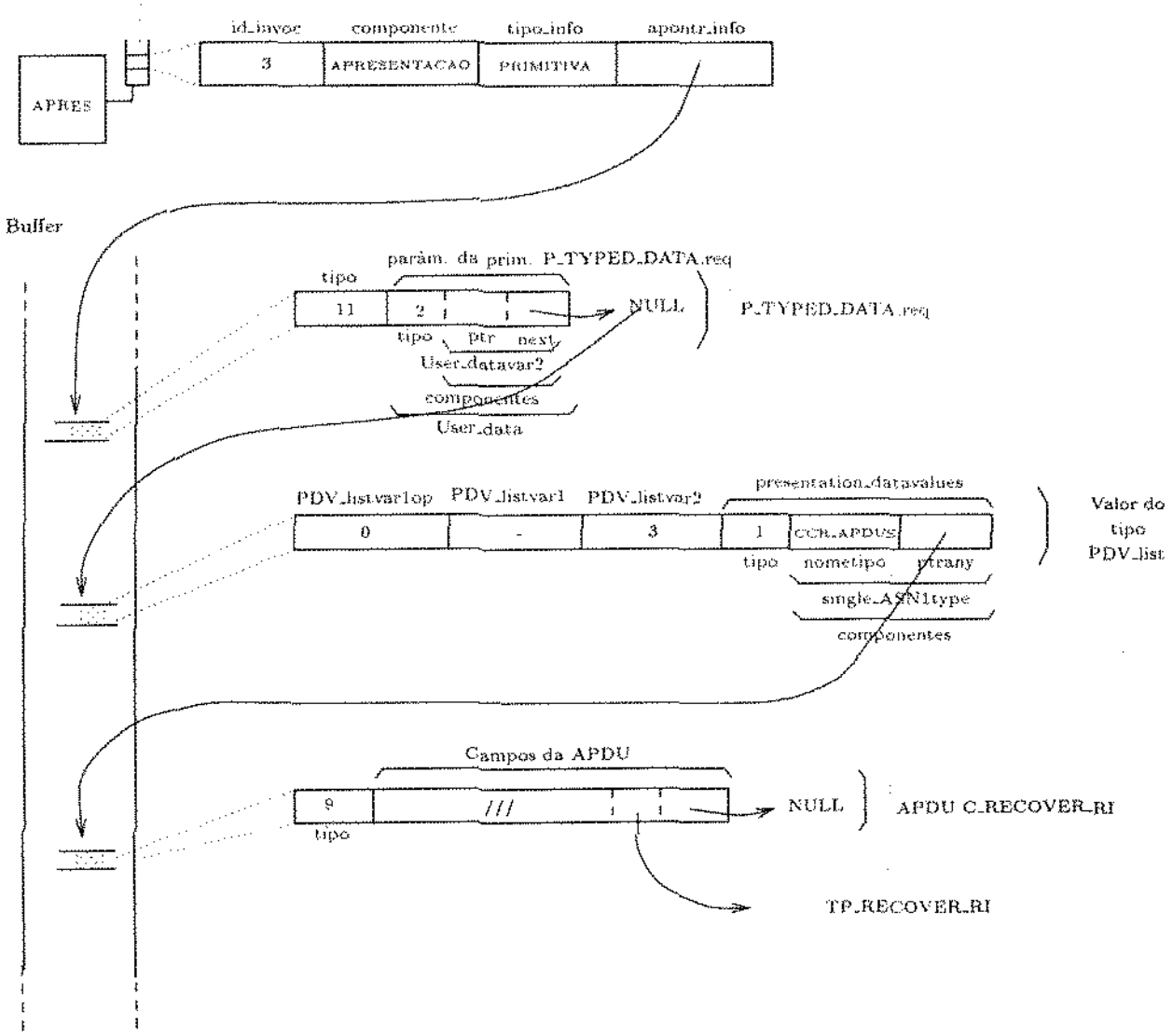

Figura 3.14: Exemplo de armazenamento de uma APDU no *buffer* para ser enviada como dado de usuário de uma primitiva da camada de Apresentação

receber esta primitiva de concatenação, o SACF vai tratá-las como se fossem eventos chegados isoladamente, mas vai concatená-las em uma mesma primitiva da camada de Apresentação.

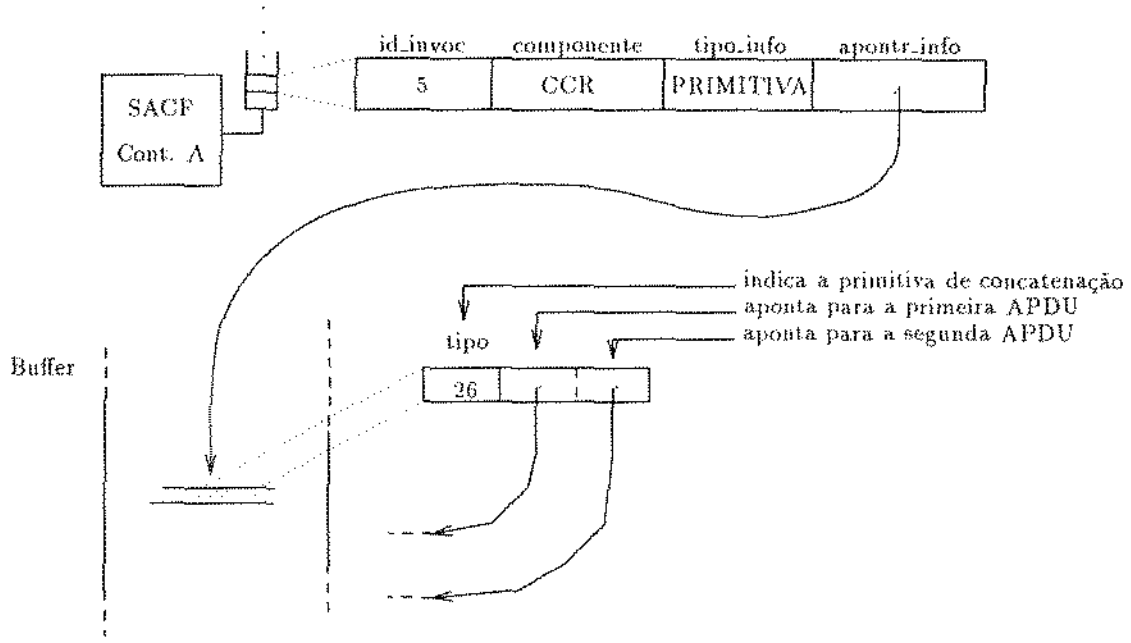

Figura 3.15: Exemplo do uso de uma primitiva de concatenação

Deve-se observar que um sistema tem que garantir que as concatenações feitas pelos ASEs juntamente com as feitas no SACF estejam de acordo com as regras do contexto de aplicação utilizado. No SISDI-OSI não estão sendo feitas concatenações genéricas, que possuem objetivo único de melhoria de desempenho. Somente estão sendo feitas as especificadas nos protocolos que trazem mais funcionalidade a eles. Como será visto no capítulo seguinte este é o caso da concatenação da APDU C-COMMIT-RI com a C-BEGIN-RI e da APDU C-ROLLBACK-RI com a C_BEGIN-RI, todas do CCR.

No envio as desalocações das APDUs são feitas pela camada de Apresentação, após a sua codificação. Na recepção de dados da rede, esta camada decodifica as APDUs e as estrutura da mesma maneira descrita neste capítulo, ou seja, seguindo-se as estruturas em C geradas pelo compilador. As APDUs são recebidas na camada de Aplicação por um SACF, como dados de usuário de primitivas da camada de Apresentação. O SACF as desconcatena, se for o caso, e as encaminha aos ASEs apropriados, que depois de usá-las as desalocarão.

O protocolo que recebe uma primitiva utiliza os seus parâmetros para a geração das PDUs. Observe que, se o mapeamento entre os campos da PDU que se for gerar e os parâmetros da primitiva for direto, basta que se troque o campo de identificação da primitiva pelo valor que identifica a PDU. A estrutura de PDUs e primitivas no *buffer* foi feita de tal maneira que esta troca pudesse ser feita, para evitar cópias de dados. Caso o mapeamento não seja direto, a PDU é montada em uma outra área, da maneira mostrada nesta seção, e a área na qual a primitiva está é desalocada. Um procedimento análogo é feito para a geração de uma primitiva a partir da recepção

de uma APDU.

Capítulo 4

O ASE CCR (Commitment, Concurrency and Recovery)

A função do CCR (*Commitment, Concurrency and Recovery*) é fornecer meios para que um conjunto de operações seja executado de forma confiável, mesmo em face de sucessivas falhas no sistema. O CCR fornecerá recursos para que os usuários de seus serviços possam iniciar e controlar uma *Ação Atômica*.

Uma ação atômica é um conjunto de operações de uma aplicação distribuída que pode ser caracterizada pelas quatro propriedades abaixo [ISO9804]:

ATOMICIDADE propriedade de um conjunto de operações relacionadas tal que as operações ou são todas executadas ou nenhuma delas é;

CONSISTÊNCIA propriedade de um conjunto de operações relacionadas tal que os efeitos das operações são precisos, corretos e válidos, no que se refere à semântica da aplicação;

ISOLAMENTO propriedade de um conjunto de operações relacionadas tal que os resultados parciais das operações não são acessíveis, exceto por operações deste conjunto; e

DURABILIDADE propriedade de um conjunto de operações relacionadas tal que nenhum efeito das operações é alterado por qualquer tipo de falha.

Por definição uma ação atômica transforma os dados sobre os quais atua de um estado consistente a outro, onde as características que um estado deve ter para ser considerado consistente dependem do tipo de aplicação que utiliza a ação atômica.

O CCR fornece primitivas para o controle de uma ação atômica em uma única associação. Uma ação atômica pode envolver, no entanto, uma ou mais de uma associação, podendo englobar vários sistemas diferentes. O usuário do CCR, utilizando este protocolo em cada uma das associações, é que fica responsável por controlar os relacionamentos entre as atividades executadas nas diversas associações, para garantir a execução correta da ação atômica como um todo. O protocolo TP, também definido pela ISO, fornece serviços para este controle da ação atômica [ISO10026-1, ISO10026-2]. Na seção seguinte será comentada uma figura que ilustra o uso destes protocolos.

Nas seções 4.1 a 4.4 seguintes é mostrado como o protocolo CCR é utilizado para se controlar uma ação atômica distribuída, apresentando-se o seu modelo geral na seção 4.1, seus serviços e primitivas na seção 4.2, como é feita a recuperação em caso de falhas, na seção 4.3, e como se comporta um tipo particular de usuário de serviços do CCR, na seção 4.4. Na seção 4.5 comenta-se sobre decisões heurísticas. Na seção 4.6 é descrito o protocolo em si, mostrando quais são os eventos que trata, qual o formato de suas APDUs, como utiliza os serviços da camada de Apresentação, dentre outros aspectos. Na seção 4.7 comenta-se como o CCR deve ser usado quando o seu usuário utilizar recursos da camada de Apresentação de forma incompatível com o uso deles pelo CCR.

4.1 O Modelo Geral do CCR

A tarefa do ASE CCR, cujos serviços e protocolo são descritos, respectivamente, em [ISO9804] e [ISO9805], é fornecer serviços para que seu usuário controle a execução de uma ação atômica sobre uma associação. O CCR não trata da natureza da ação atômica, ou seja, de que tipos de operações constituem a ação. A natureza da ação é dependente do usuário do CCR. Por exemplo, o protocolo FTAM (*File Transfer, Access and Management*) [ISO8571] pode usar os serviços do CCR para garantir atomicidade em operações sobre arquivos, enquanto o RDA [ISO9579-1] pode usar os seus serviços para executar operações atômicas sobre bancos de dados. As primitivas do CCR servem para delimitar o início e controlar o fim de uma ação atômica em uma associação, independentemente de qual for a natureza desta ação. Em uma ação atômica que envolva mais de uma associação pode-se ter um tipo de protocolo diferente como usuário do CCR em cada uma delas.

Quando dois usuários de serviços do CCR se comunicam para a realização de uma ação atômica, a relação de cooperação que há entre eles é chamada de *Ramo da Ação Atômica*. Em um dado ramo da ação atômica, o usuário que o inicia é chamado de *Usuário Superior do Ramo da Ação Atômica*, ou, simplesmente, *SUPERIOR*. O outro usuário, a quem o *SUPERIOR* enviou o pedido de início do ramo, é chamado de *Usuário Subordinado do Ramo da Ação Atômica*, ou, simplesmente, *SUBORDINADO*. Para suportar as trocas de dados em um ramo da ação atômica entre usuários do CCR em sistemas diferentes, uma associação deve ser previamente estabelecida. Não há, no entanto, relacionamento entre o usuário estabelecedor da associação e a função que este desempenha no ramo de uma ação atômica, ou seja, um usuário pode ser o estabelecedor da associação e exercer função de superior ou subordinado.

Um usuário do CCR pode ser ao mesmo tempo subordinado em relação a um ramo da ação atômica e superior em relação a outros. O protocolo restringe, no entanto, que estes relacionamentos entre os usuários de serviços do CCR que participam de uma mesma ação atômica formem uma árvore. Nesta árvore os nós são constituídos por usuários do CCR e as arestas por ramos da ação atômica. Esta árvore é chamada de *Árvore da Ação Atômica* e está ilustrada na figura 4.1. Os serviços do CCR, apesar de considerarem a existência desta árvore, são especificados para serem usados em apenas um ramo. Para o controle da ação atômica como um todo existe o protocolo TP [ISO10026-1, ISO10026-2, ISO10026-3]. Este protocolo fornece serviços para o controle da árvore de modo global e para isto utiliza os serviços do CCR em cada ramo. [Joh] mostra um exemplo de como os protocolos TP e CCR são usados conjuntamente para fornecer transações atômicas para uma aplicação, no caso, acesso a bancos de dados remotos, através do protocolo RDA (embora neste artigo é feita referência à versão do TP de 1990, anterior à referenciada nesta dissertação).

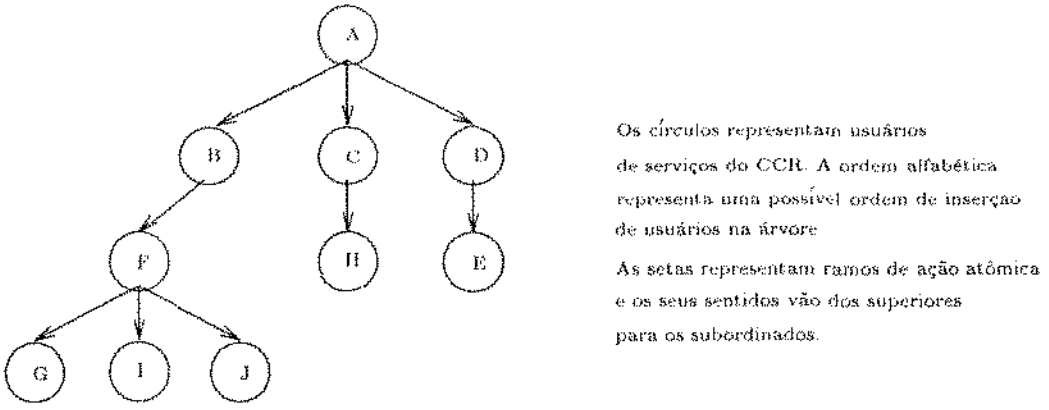

Figura 4.1: Árvore de ação atômica

Definindo-se mais especificamente, um *usuário do CCR* (ou *usuário de serviços do CCR*) corresponde à parte de uma API (*Application Process Invocation*) e de sua AEI (*Application Entity Invocation*) que utilizam os serviços do CCR para coordenar um ou mais ramos relacionados de uma árvore de ação atômica. Uma AEI e sua API podem suportar mais de um usuário do CCR de uma mesma ação atômica ou de diferentes ações atômicas. Os usuários A e J da figura 4.1, por exemplo, poderiam ser suportados por uma mesma AEI.

O usuário do CCR que cria uma árvore de ação atômica, ou seja, o superior do primeiro ramo desta árvore, é chamado de *MESTRE*. Ele será o responsável pela decisão sobre o resultado final da ação atômica como um todo. Uma ação atômica é identificada de acordo com o seu MESTRE, como será visto posteriormente.

Para a correta operação do CCR, há dois tipos de dados que devem ser mantidos de forma segura, ou seja, devem ser tratados de forma tal que, mesmo em casos de falhas, estejam novamente disponíveis para a AEI e API que os usarem, após procedimentos de recuperação. Caso estes dados sejam perdidos, devido a uma falha, por exemplo, perde-se a aplicabilidade do protocolo CCR, não podendo garantir-se mais a manutenção das propriedades da ação atômica. Os tipos são:

1. os dados da própria aplicação, chamados de *dados ligados* (*bound data*), que constituem os dados sobre os quais as operações da ação atômica atuam; e

2. os dados de controle e informações de estado necessários para recuperação de uma ação atômica e seus ramos. Estes dados vão constituir parte dos chamados *dados da ação atômica*. A expressão *dados da ação atômica* se refere a quaisquer dados de controle e estado de uma ação atômica e seus ramos, não só àqueles necessários a recuperação.

Aos dados ligados estão associados dois estados: um *estado inicial*, que é o conjunto dos valores que os dados ligados possuem antes de qualquer operação da ação atômica ser executada, e um *estado final*, que é o conjunto dos valores que eles devem possuir após uma execução, sem falhas, das operações que constituem a ação atômica.

O CCR vai dar suporte à efetivação e à recuperação de ações atômicas através do fornecimento de meios para se executar o protocolo *Presumed Rollback* (ou *Presumed Abort*), apresentado em

[Moh83]. O protocolo *Presumed Rollback* é uma otimização do protocolo *Two-Phase Commit* (*Commit em duas fases*) básico [Ber87, Moh83], bastante conhecido na área de bancos de dados.

O ASE CCR é uma especificação deste protocolo para o RM-OSI, com uma pequena variação, comentada na seção 4.6.1, causada por características deste modelo. O *Presumed Rollback* assume que os componentes que fazem parte da ação atômica formam uma estrutura de árvore, motivo pelo qual os usuários do CCR tinham que formar a árvore da ação atômica. Nesta árvore somente há comunicação entre pai e filho. Embora o *Presumed Rollback* atue em toda a árvore, ressalta-se novamente que os serviços do CCR foram definidos para uso em uma única associação. O controle dos relacionamentos entre as atividades nas diversas associações para a execução do *Presumed Rollback* fica a cargo do usuário do CCR, e os serviços do protocolo TP são definidos para este fim.

As trocas de informações entre os usuários, segundo o protocolo *Presumed Rollback*, para que todos os usuários terminem a ação atômica de maneira consistente, em uma situação em que não há falhas, são as seguintes :

1. o superior informa a cada um dos subordinados qual é a tarefa que devem realizar (figura 4.2a);

2. cada subordinado analisa as tarefas e informa ao superior se pode realizá-las ou não. Se um subordinado respondeu que pode realizar as tarefas, ele passa a esperar uma resposta do superior, indicando se deve executá-las ou não. Se o subordinado responder que não pode realizá-las, a ação atômica terá terminado para este subordinado (a figura 4.2b1 ilustra a situação em que todos os subordinados comunicaram que podem realizar as tarefas e a figura 4.2b2 ilustra a situação em que um dos subordinados indica que não pode realizá-las);

3. o superior, se receber uma resposta afirmativa de *TODOS* os subordinados, ordena então a cada um deles que realizem as tarefas (figura 4.2c1). Os subordinados então efetivam as tarefas, colocando os dados em seu estado final, e confirmam o término de execução ao superior (figura 4.2d). Caso algum dos subordinados tenha indicado que não pode realizar o trabalho, o superior, então, ordena a não realização das tarefas aos subordinados que aguardam uma resposta sua, os seja, os que haviam respondido afirmativamente (figura 4.2c2). Estes subordinados podem, a seguir, apagar quaisquer informações que possuam a respeito desta ação atômica e restaurar os dados ao estado inicial, não precisando enviar confirmações. No primeiro caso diz-se que o superior ordenou *COMMIT*, ou seja, a efetivação da operação, enquanto no segundo caso diz-se que o superior ordenou *ROLLBACK*, ou seja, a volta ao estado anterior ao primeiro passo na execução da ação atômica. No caso de *COMMIT*, após os dados atingirem o estado final, não há mais meios de se voltar a um estado anterior dentro desta ação atômica.

Observe que, executando-se os passos acima, todos os componentes da ação atômica a terminarão da mesma maneira, ou executando *COMMIT* ou *ROLLBACK*, fazendo com que todos os dados manipulados pela ação fiquem consistentes em todos os sistemas. Se todos os subordinados podem executar as tarefas, levando os dados para o estado final, então este será o resultado da ação. Caso pelo menos um deles não possa, então todos os usuários colocarão os dados no estado inicial, fazendo com que tudo se passe como se a ação atômica não tivesse sido executada. As ações que devem ser feitas para garantir um final consistente para a ação atômica em casos em que ocorrem falhas serão comentadas na seção 4.3.

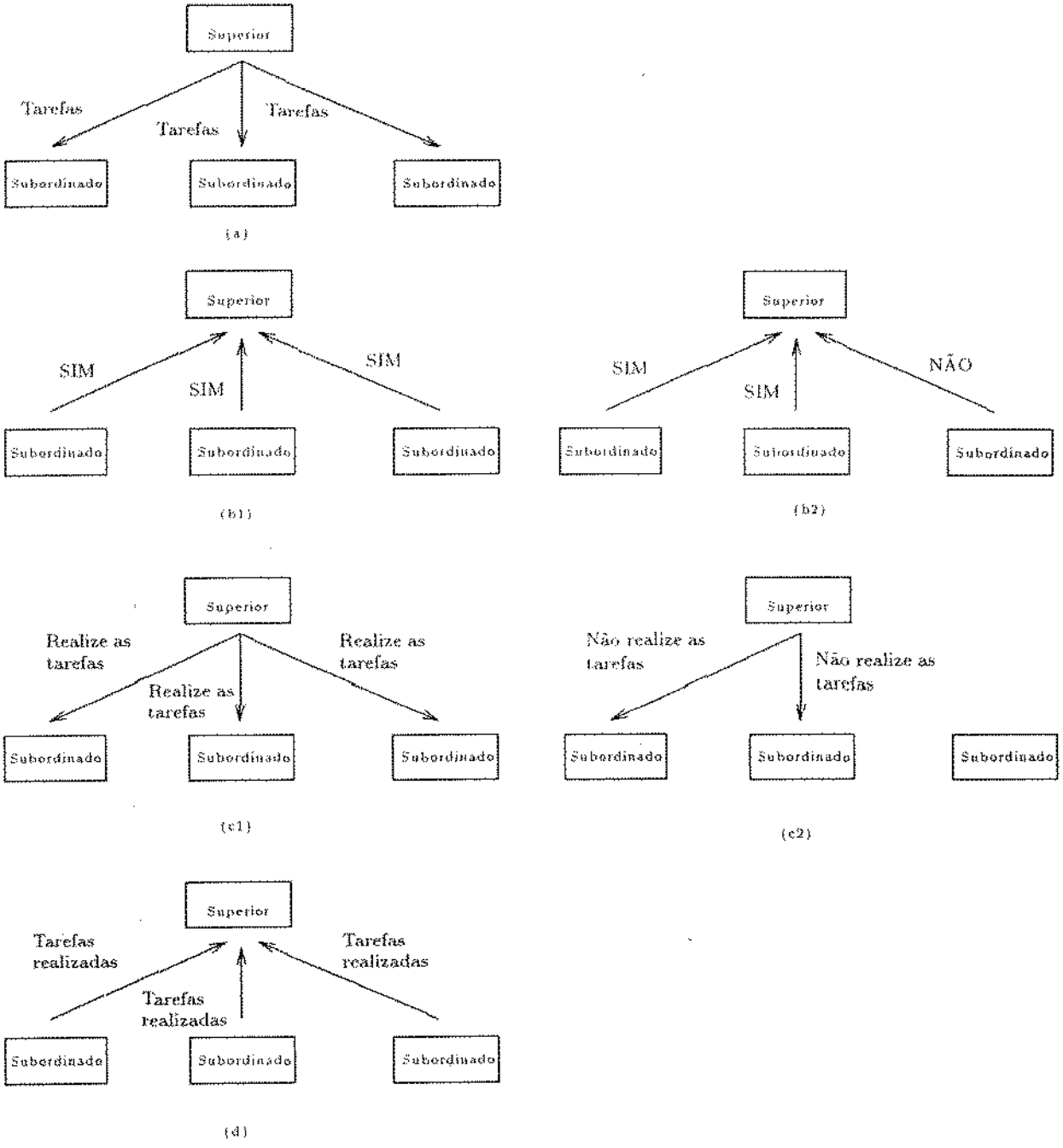

Figura 4.2: O protocolo de duas fases

Na fase 2 acima devem-se tomar as medidas necessárias para garantir o controle de concorrência entre diferentes ações atômicas. Se o mecanismo utilizado para isto for *locks*, que é um mecanismo tradicional para a obtenção de exclusão mútua em dados compartilhados [Ber87], então nesta fase deve-se dar o *lock* nos dados que serão utilizados pela ação atômica. Na fase 3, após o *COMMIT* ou o *ROLLBACK*, ou na própria fase 2, caso o subordinado responda que não pode realizar as tarefas, todos os recursos utilizados pela ação atômica devem ser liberados. Isto inclui a liberação de todos os *locks* usados.

Os serviços do CCR servem justamente para que as fases acima sejam determinadas e realizadas e para tratar de recuperação em caso de falhas. O fornecimento dos meios efetivos para se garantir o controle de concorrência e a manutenção de dados seguros é responsabilidade do usuário do CCR, que pode escolher a maneira de fornecer estas garantias que melhor se adapte à sua aplicação. O CCR, como todos os outros protocolos do RM-OSI, trata apenas dos aspectos de comunicação envolvidos nos problemas de cooperação entre APs, não tratando, portanto, de aspectos que sejam puramente de processamento de informação das aplicações. O seu funcionamento independe da maneira escolhida pelo usuário para o fornecimento de tais recursos.

O protocolo é de duas fases pois pode-se considerá-lo como dividido em uma primeira etapa, onde o superior informa as tarefas e cada subordinado responde se pode ou não executá-las, e em uma segunda etapa seguinte, onde o superior, em posse das respostas dos subordinados, decide ordenar *COMMIT* ou *ROLLBACK*. A razão do nome *Presumed Rollback* será explicada na seção 4.3.

4.2 Os Serviços e suas Primitivas

O CCR fornece serviços confirmados, não confirmados e um opcionalmente confirmado. No primeiro caso o serviço possui primitivas dos tipos *request*, *indication*, *response* e *confirm*; no segundo, apenas dos tipos *request* e *indication*; no terceiro, possui os quatro tipos de primitivas quando houver a confirmação, e só os dois primeiros, em caso contrário (a emissão em um sistema de uma primitiva do tipo *request* ou *response* causa a emissão no sistema remoto de, respectivamente, uma primitiva do tipo *indication* ou *confirm*). Se um receptor irá confirmar ou não o serviço opcionalmente confirmado depende do atendimento a certas condições. As primitivas serão indicadas neste texto pelo nome do serviço seguido de um sufixo que indica o tipo da primitiva. Os sufixos usados são *req*, *ind*, *rsp* e *cnf*, respectivamente, para as primitivas dos tipos *request*, *indication*, *response* e *confirm*.

Os serviços fornecidos pelo CCR são:

C-BEGIN utilizado por um superior, inicia um ramo de uma ação atômica. Nenhum outro ramo de ação atômica pode estar ativo na associação. É opcionalmente confirmado. As condições nas quais este serviço será ou não confirmado serão comentadas na seção 4.6.1.

As primitivas C-BEGIN.req e C-BEGIN.ind possuem os seguintes parâmetros:

identificador da ação atômica é um parâmetro obrigatório, formado pelo *nome*, conforme [ISO7498-3], do mestre da ação atômica, e por um sufixo, usado para diferenciar as ações atômicas de um mesmo mestre. O nome do mestre é o título da entidade de aplicação que representa o MESTRE. Este nome identifica esta entidade de forma não ambígua, em toda a rede;

identificador do ramo da ação atômica também é um parâmetro obrigatório, e é formado pelo *nome*, conforme [ISO7498-3], do superior deste ramo, e por um sufixo, utilizado para diferenciar cada ramo de uma mesma ação atômica com mesmo superior. O nome do superior é o título da AE que o representa, e este nome identifica esta AE, de forma não ambígua, em toda a rede;

dados do usuário parâmetro opcional, utilizado para transportar qualquer informação que o usuário queira. O CCR não o interpreta.

As primitivas C-BEGIN.rsp e C-BEGIN.cnf possuem somente um campo de dados do usuário, com mesma utilidade do campo de mesmo nome das primitivas acima. Também é um parâmetro opcional.

C-PREPARE utilizado por um superior, indica o fim das transmissões, por parte do superior, de mensagens que alterem o estado dos dados ligados e solicita ao subordinado que emita ou uma primitiva C-READY.req ou C-ROLLBACK.req (descritas abaixo). Mesmo após receber a primitiva C-PREPARE.ind o subordinado ainda pode enviar para o superior mensagens que fazem parte da ação atômica e que podem alterar o estado dos dados ligados. É um serviço não confirmado e corresponde, na descrição do protocolo *Presumed Rollback* feita anteriormente, a solicitar que o subordinado responda se pode ou não realizar as tarefas.

As primitivas deste serviço, C-PREPARE.req e C-PREPARE.ind, possuem somente um campo para dados do usuário, opcional, com a mesma utilidade do campo de mesmo nome das primitivas anteriores.

Este serviço é opcional. Se, por características da aplicação que estiver usando o CCR, o subordinado souber que o superior não vai mais enviar dados e passará a esperar a indicação de que as tarefas podem ou não ser realizadas, este serviço não precisa ser utilizado.

C-READY utilizado por um subordinado, serve para que este indique ao superior que as tarefas solicitadas poderão ser realizadas. Diz-se que se está *oferecendo efetivação*. É não confirmado.

As primitivas deste serviço, C-READY.req e C-READY.ind, possuem somente um campo para dados do usuário, opcional, com a mesma utilidade do campo de mesmo nome das primitivas anteriores.

Como mencionado anteriormente o serviço C-PREPARE é opcional. Portanto o subordinado não precisa esperar a chegada de uma primitiva C-PREPARE.ind para emitir uma primitiva C-READY.req.

Antes da emissão desta primitiva o subordinado deve ter todos os dados necessários para recuperação armazenados de forma segura (a seção 4.3 comentará sobre recuperação). Os dados ligados têm também que estar gravados de forma segura, juntamente com informações suficientes de modo que possam ser colocados na estado inicial, caso o resultado da ação atômica seja *ROLLBACK*, ou no estado final, caso o resultado seja *COMMIT*.

Após emitir esta primitiva ao superior, o subordinado não pode mais enviar mensagens que alterem o estado dos dados ligados.

C-COMMIT utilizado por um superior, serve para que este ordene a efetivação aos subordinados, ou seja, ordene que as tarefas sejam realizadas e, com isto, os dados atinjam o estado final. É confirmado.

As primitivas deste serviço, C-COMMIT.req, C-COMMIT.ind, C-COMMIT.rsp e C-COMMIT.cnf, possuem somente um campo para dados do usuário, opcional, com a mesma utilidade do campo de mesmo nome das primitivas anteriores.

Antes de emitir a primitiva C-COMMIT.req, o superior tem que ter seus dados ligados mantidos de forma que possam ser colocados no estado final e dados utilizados para recuperação dos ramos com os subordinados têm que ser armazenados de forma segura (a seção 4.3 comentará sobre recuperação). Posteriormente, mas não necessariamente antes da emissão, os dados têm que ser efetivamente colocados neste estado e os recursos utilizados devem ser liberados. O subordinado, antes de enviar a primitiva C-COMMIT.rsp, tem que colocar seus dados no estado final e liberar os recursos utilizados. Assim que emitir esta primitiva a ação terá acabado para o subordinado. Para o superior, ela terminará assim que receber as primitivas C-COMMIT.cnf de todos os seus subordinados.

C-ROLLBACK utilizado por um superior ou por um subordinado. No primeiro caso, é utilizado para que o superior ordene *ROLLBACK* aos subordinados, ou seja, ordene que os dados ligados sejam restaurados ao seu estado inicial; no segundo caso, é utilizado para que o subordinado informe ao superior que ele não irá mais prosseguir com o ramo da ação atômica. Em particular serve para que o subordinado *negue efetivação*, ou seja, indique que as tarefas não poderão ser realizadas. É confirmado.

As primitivas deste serviço, C-ROLLBACK.req, C-ROLLBACK.ind, C-ROLLBACK.rsp e C-ROLLBACK.cnf, possuem somente um campo para dados do usuário, opcional, com a mesma utilidade do campo de mesmo nome das primitivas anteriores.

Uma primitiva C-ROLLBACK.req pode ser emitida por um superior a qualquer momento antes de enviar C-COMMIT.req, e isto pode ser feito mesmo que todos os subordinados tenham dito que podem realizar as tarefas (isto pode acontecer, por exemplo, se o próprio superior tiver operações a fazer como parte da ação atômica e não puder realizá-las). Obviamente se o superior decidir por emitir C-ROLLBACK.req, ele não pode mais emitir C-COMMIT.req. Um subordinado pode emitir C-ROLLBACK.req a qualquer momento antes de emitir C-READY.req, pois uma vez emitida esta última primitiva, ele deve esperar o resultado da ação atômica. Caso ele emitisse C-ROLLBACK.req depois de C-READY.req, poderia ocorrer de o superior já ter recebido todas as primitivas C-READY.ind, ter decidido por *COMMIT* e já ter informado esta decisão a alguns subordinados, antes de receber a primitiva C-ROLLBACK.ind. Esta situação criaria inconsistência entre os subordinados;

e

C-RECOVER utilizado por um superior ou por um subordinado. É usado por quem possui *responsabilidade de recuperação* para efetuar a ressincronização do ramo da ação atômica, após uma falha de aplicação ou de comunicação, colocando os dados em um estado consistente. O subordinado adquire responsabilidade de recuperação quando oferece efetivação e o superior, quando decide ordenar efetivação (o processo de recuperação será mostrado na seção 4.3). Quando usado no sentido do superior para o subordinado, é um serviço confirmado; no sentido inverso é opcionalmente confirmado.

As primitivas deste serviço, C-RECOVER.req, C-RECOVER.ind, C-RECOVER.rsp e C-RECOVER.cnf, possuem os seguintes parâmetros:

estado de recuperação é um parâmetro obrigatório e é utilizado para fornecer informações sobre o estado do usuário do CCR, ao se restabelecer um ramo da ação atômica, durante procedimentos de recuperação de falhas. Quando a primitiva é emitida do superior para o subordinado, este parâmetro pode possuir os seguintes valores:

- se for uma primitiva do tipo *request*: valor : *commit*;
- se for uma primitiva do tipo *response*: valores: *unknown* ou *retry-later*.

Quando a primitiva é emitida do subordinado para o superior, este parâmetro pode possuir os seguintes valores:

- se for uma primitiva do tipo *request*: valor: *ready*;
- se for uma primitiva do tipo *response*: valores: *done* ou *retry-later*;

A seção seguinte descreverá o uso deste parâmetro;

identificador da ação atômica parâmetro obrigatório, identifica a ação atômica à qual pertence o ramo que se está recuperando. É formado da mesma maneira que o parâmetro de mesmo nome das primitivas do serviço C-BEGIN;

identificador do ramo da ação atômica parâmetro obrigatório, identifica o ramo da ação atômica que se está recuperando. É formado da mesma maneira que o parâmetro de mesmo nome das primitivas do serviço C-BEGIN;

dados do usuário parâmetro opcional, possui a mesma utilidade do parâmetro de mesmo nome das primitivas descritas anteriormente.

O CCR permite que as primitivas que formam cada par abaixo

- C-COMMIT.req e C-BEGIN.req e
- C-ROLLBACK.req e C-BEGIN.req

sejam emitidas juntas. Feito desta maneira o usuário pode indicar o resultado de uma ação atômica e já iniciar uma nova. Estes pares de primitivas só podem ser emitidas juntas por um superior.

Supondo-se uma situação com um superior e n subordinados, o diagrama de tempo da figura 4.3 ilustra uma seqüência normal de operação em que há efetivação da ação atômica (decide-se por *COMMIT*). Nesta figura o bloco da esquerda representa o superior, o bloco da direita, os subordinados, e a lacuna central representa o provedor do serviço. Os números entre parênteses servem para indicar os ramos da ação atômica.

As primitivas do serviço C-BEGIN iniciam os ramos da ação atômica, um para cada subordinado. A seguir são transmitidas primitivas que são específicas da aplicação que está fazendo uso do CCR, por exemplo, poderiam ser emitidas neste período primitivas do RDA, em uma ação atômica constituída de operações de acesso a bancos de dados remotos. As primitivas do serviço C-PREPARE estão indicando que as operações da aplicação que está utilizando o CCR terminaram nos ramos da ação atômica, para o superior, e, portanto, passa a esperar que os subordinados indiquem se oferecem ou negam efetivação. Neste exemplo todos os subordinados ofereceram efetivação. Em seguida a ordem de *COMMIT* é dada pelo superior a cada um dos subordinados. Somente se todos os subordinados oferecerem efetivação é que o superior ordenará o *COMMIT*.

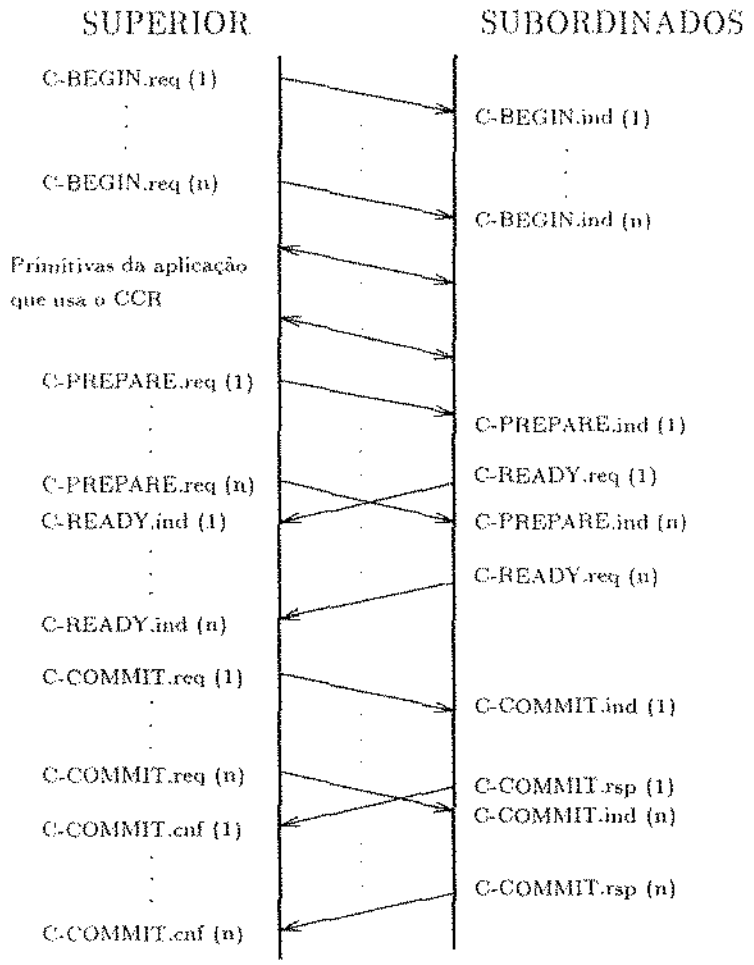

Figura 4.3: Diagrama de tempo para um caso em que ocorre efetivação

Deve-se notar nesta figura que há uma instância do protocolo CCR para cada associação. O controle dos resultados em várias associações é feito pelo usuário destas várias instâncias do CCR. Por exemplo, é este usuário que se encarrega de verificar se todos os subordinados emitiram uma primitiva C-READY.req. Quando o CCR é usado com o protocolo TP, é o TP quem faz este controle.

Um diagrama de tempo para a mesma situação anterior, mas em que não há efetivação é mostrado na figura 4.4, onde os significados das colunas, da lacuna e dos valores entre parênteses são os mesmos da figura 4.3. Como no diagrama desta figura, as primitivas do serviço C-BEGIN iniciam o ramo da ação atômica. As primitivas do serviço C-PREPARE são novamente usadas para informar o término, por parte do superior, das operações da aplicação que está utilizando o CCR nos ramos da ação atômica. Os subordinados nas associações 1 a (n-1) estão oferecendo efetivação, enquanto o subordinado na associação n está negando, através do serviço C-ROLLBACK. A seguir o superior ordena *ROLLBACK* a todos os subordinados que ofereceram efetivação e confirma a primitiva do subordinado que negou. A seguir os subordinados nos ramos (1) a (n-1) confirmam o serviço C-ROLLBACK. Com isto os dados voltarão ao seu estado inicial no superior e em todos os subordinados. Basta que um subordinado negue efetivação para que o superior necessariamente ordene *ROLLBACK*.

Como os serviços do CCR são definidos para uma só associação, nos diagramas anteriores, para cada associação há uma invocação distinta do CCR. O que os exemplos querem mostrar é como o CCR é utilizado por um usuário, em cada associação, para realizar o protocolo de duas fases.

A seção a seguir mostra como é feita a recuperação de um ramo da ação atômica.

4.3 Recuperação

Após a ocorrência de uma falha de aplicação ou de comunicação, procedimentos de recuperação devem ser executados para que um participante de uma ação atômica restaure os dados ligados sob sua responsabilidade a um estado consistente.

Como comentado no início deste capítulo, pela própria definição de ação atômica, os dados sobre os quais as suas operações atuam só estarão em estados consistentes logo antes da ação começar (estado inicial) e logo depois de seu término (estado final). As operações que formam uma ação atômica, tomadas isoladamente, podem levar estes dados a um estado intermediário inconsistente. Portanto os procedimentos de recuperação de uma falha devem restaurar os dados ou para o estado inicial ou para o estado final, de maneira global, ou seja, em todos os sistemas que participam da ação.

Uma falha de aplicação é qualquer problema que impeça uma invocação de Entidade de Aplicação de atingir seus objetivos especificados. Por exemplo, um *crash* no sistema. Uma falha de comunicação é qualquer problema que resulte no rompimento de uma associação de aplicação que esteja suportando um ramo da ação atômica. Por exemplo, um erro de protocolo detectado em alguma camada inferior.

O serviço C-RECOVER é utilizado para que superior e subordinado se sincronizem novamente em um estado consistente após uma falha. Esta sincronização só pode ser iniciada por aquele usuário que possuir *responsabilidade de recuperação*.

Como mencionado anteriormente, um subordinado adquire responsabilidade de recuperação quando oferece efetivação, ou seja, quando for emitir uma primitiva C-READY.req. Um superior

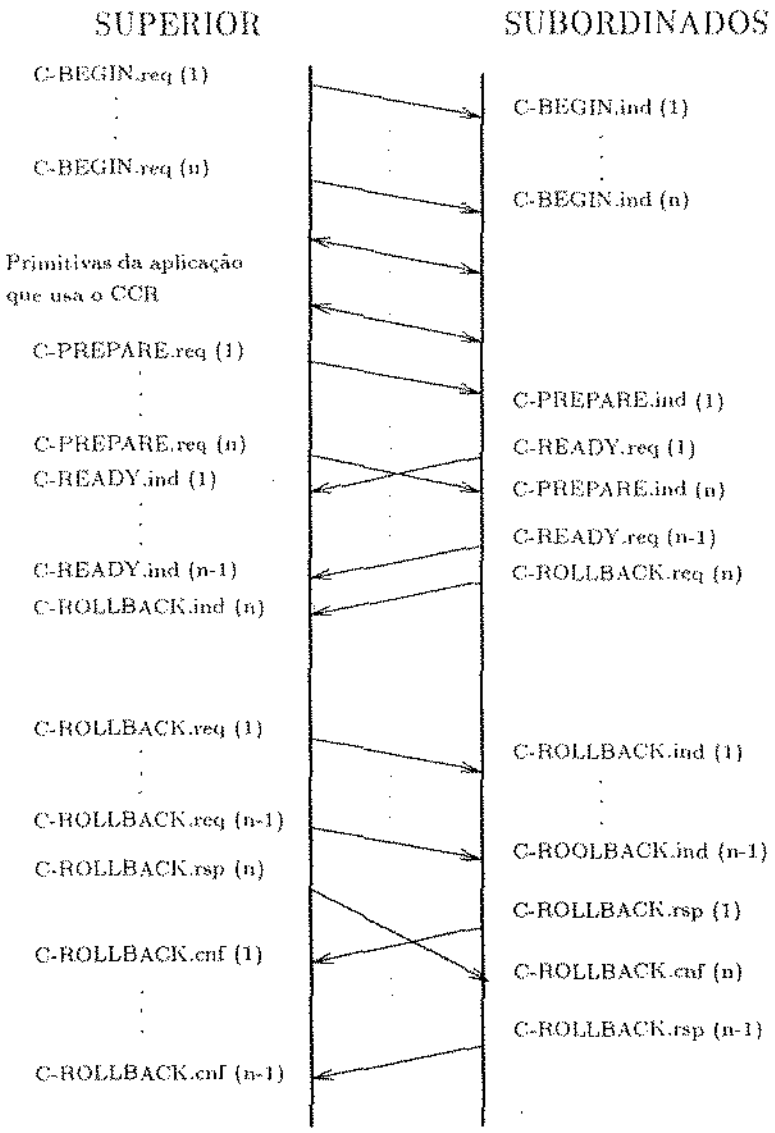

Figura 4.4: Diagrama de tempo para um caso em que ocorre *rollback*

a adquire quando decide ordenar efetivação, ou seja, quando for emitir C-COMMIT.req. Uma vez adquiridas, as responsabilidades de recuperação só terminam quando a ação atômica terminar.

Os usuários utilizam-se de *registros de log*, que são registros de informações mantidos de forma segura. Estes registros vão constituir parte dos dados da ação atômica, e são usados para se determinar se um usuário possui ou não responsabilidade de recuperação e para armazenar informações utilizadas no processo de recuperação. Os registros de *log* usados pelos usuários do CCR são dois:

registro de oferecimento de efetivação gravado por um subordinado quando decide oferecer efetivação. De fato considera-se que um subordinado ofereceu efetivação e, portanto, passa a ter responsabilidade de recuperação, tão logo este registro seja gravado, e não exatamente no momento de emissão da primitiva C-READY.req. Este registro contém informações necessárias à recuperação do ramo com o superior, como, por exemplo, o título de AE, o identificador de invocação de AE e o endereço de apresentação do superior, dentre outras. Este registro é armazenado antes de se emitir a primitiva C-READY.req, após os dados estarem de modo que possam ser colocados no estado inicial ou final;

registro de ordenação de efetivação gravado por um superior quando decide ordenar efetivação. De modo análogo ao caso anterior, considera-se que um superior ordenou efetivação e, portanto, passa a ter responsabilidade de recuperação, tão logo este registro seja gravado, e não exatamente quando a primitiva C-COMMIT.req for emitida. Este registro é gravado antes de esta primitiva ser emitida e após os dados estarem de forma que possam ser colocados no estado final.

O controle do armazenamento destes registros, incluindo a garantia de que este armazenamento é feito de forma segura, é feito pelo usuário do CCR.

O protocolo no qual o CCR se baseia, como mencionado anteriormente, possui o nome de *Presumed Rollback* (*ROLLBACK* pressuposto). Este nome vem do fato de um usuário do CCR assumir que houve *ROLLBACK* em uma ramo da ação atômica se, para este ramo, após a recuperação de uma falha de aplicação ou de comunicação, uma das duas situações seguintes ocorrer:

1. este usuário não possuir nenhum dado da ação atômica utilizado para recuperação;
2. o usuário do CCR possuir dados da ação atômica utilizados para recuperação, mas seu superior não.

As situações acima indicam que, durante a fase de recuperação da ação atômica, pressupõe-se que o resultado da ação atômica foi *ROLLBACK* quando não houver informação sobre o resultado da ação.

Nos parágrafos seguintes se descreve mais detalhadamente o processo de recuperação, assumindo-se novamente uma situação com um superior e alguns subordinados.

Para se processar a recuperação de uma falha, uma nova associação é utilizada. Esta associação não pode estar sendo usada para suportar outro ramo ou qualquer outra recuperação de ramo. O processo de recuperação ocorre da seguinte maneira:

1. se iniciado pelo subordinado: dois casos podem ocorrer:

(a) se, ao recuperar, o subordinado não possuir um registro de oferecimento de efetivação gravado, isto indica que ainda não ofereceu efetivação. Portanto seu superior não poderia ter decidido por *COMMIT* e, com isto, também não possui dados da ação atômica. Como o protocolo de recuperação determina que, na ausência de dados da ação atômica para recuperação, o usuário deve executar *ROLLBACK*, o subordinado pode restaurar os dados ao estado inicial, pois o superior fará o mesmo, e se atingirá um estado consistente dos dados;

(b) se possuir um registro, quer dizer que foi oferecida efetivação e o superior pode, portanto, ter decidido por *COMMIT* ou *ROLLBACK* ou não se ter decidido ainda. O subordinado então envia uma primitiva C-RECOVER(ready).req (o valor entre parêntes representa o valor do parâmetro *estado de recuperação* da primitiva. Esta convenção será usada outras vezes neste texto). Se o superior não tiver um registro armazenado (significando ou que havia decidido *ROLLBACK*, pois neste caso não se grava registro de *log*, ou que não havia ainda decidido o resultado da ação), este responde com C-RECOVER(unknown).rsp. O subordinado com isto retorna os dados ao estado inicial (figura 4.5a). Se, por outro lado, o superior tiver um registro, significa que havia ordenado *COMMIT*. Nesta situação o superior não responde ao serviço anterior e invoca C-RECOVER(commit).req. O subordinado, ao receber a indicação desta primitiva, coloca os seus dados no estado final e responde com C-RECOVER(done).rsp (figura 4.5b). Observe que é por isto que o serviço C-RECOVER é opcionalmente confirmado quando emitido pelo subordinado;

2. se iniciado pelo superior: novamente dois casos existem:

(a) se, ao recuperar, não possuir um registro de ordenação de efetivação, significa que não havia decidido por *COMMIT*. O superior então decide-se por *ROLLBACK*, mesmo que todos os subordinados tenham indicado que podiam realizar as tarefas (observe que após uma falha, se ainda não tiver decidido por *COMMIT*, o superior não sabe se os subordinados ofereceram ou não efetivação, pois não armazena informações sobre isto). Os seus dados são colocados no estado inicial e os dados da ação atômica são apagados (diz-se que o superior "esquece" a ação atômica). Pelo ítem 1 acima, vê-se que nesta situação os subordinados também decidirão por *ROLLBACK* e os dados ficarão em um estado consistente;

(b) se, ao recuperar, possuir um registro de ordenação de efetivação, o superior deve transmitir o resultado aos subordinados, enviando C-RECOVER(commit).req. Se o superior decidiu efetivação é porque os seus subordinados a haviam oferecido anteriormente. Portanto os subordinados colocam seus dados no estado final e respondem com C-RECOVER(done).rsp (figura 4.5c).

O valor *retry-later* para o parâmetro *estado de recuperação* é usado em situações em que o superior ou o subordinado não podem atender às solicitações de recuperação no momento que as receberam, por fatores transientes. Basta que o emissor da solicitação a envie novamente algum tempo depois.

O processo de recuperação de ações atômicas pode ser iniciado por qualquer um, superior ou subordinado, após uma falha, assim que o sistema puder continuar suas operações. Basta que o usuário possua responsabilidade de recuperação. Como será visto posteriormente, mesmo quando

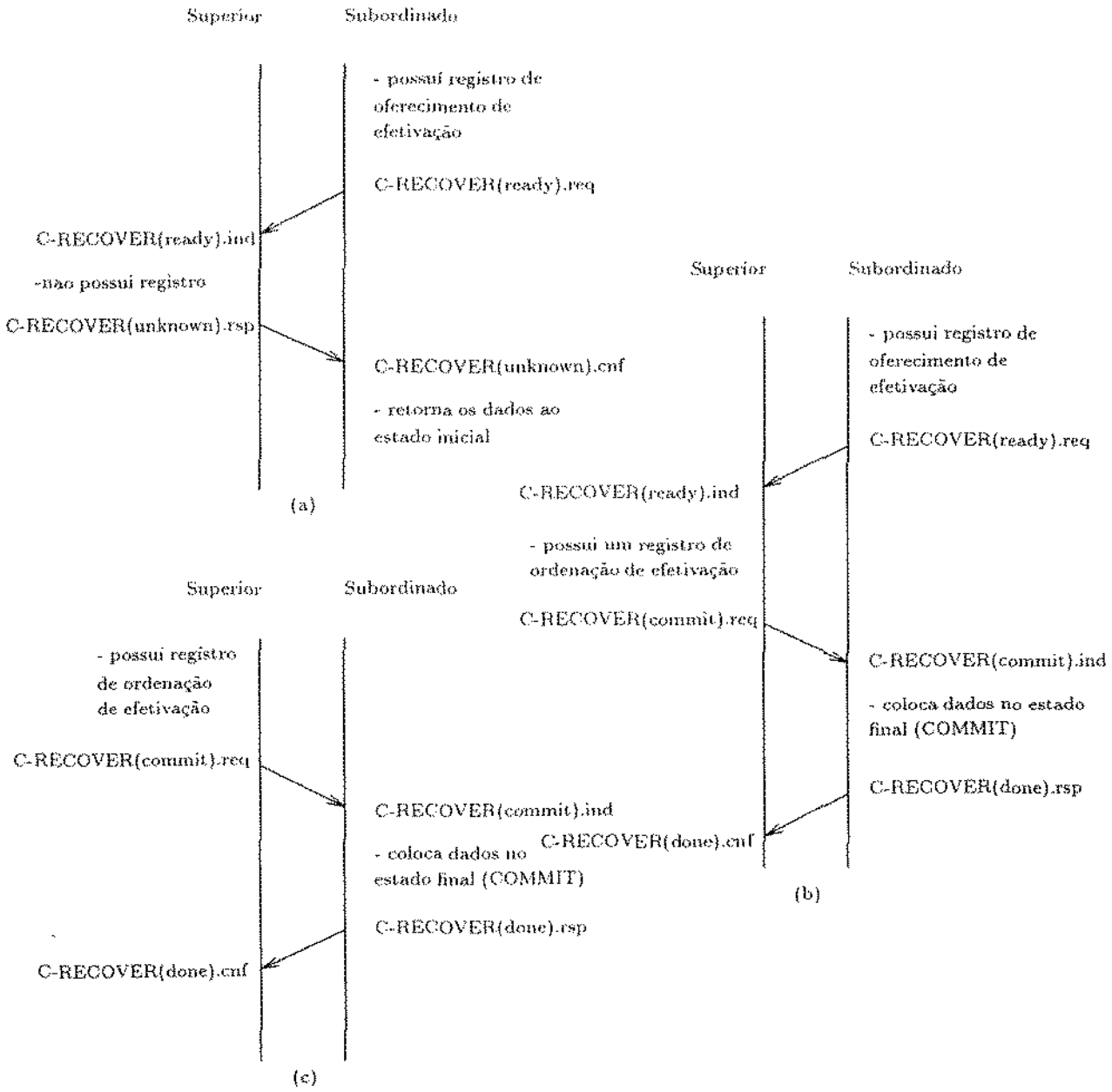

Figura 4.5: Casos de recuperação

ambos, superior e subordinado, tiverem responsabilidade de recuperação, em um dado instante somente um deles poderá emitir a primitiva C-RECOVER.req (ou seja, não há *colisão* destas primitivas).

Observe que um subordinado adquire responsabilidade de recuperação a partir do momento em que não pode decidir unilateralmente por *ROLLBACK* e tem que interagir com o superior para saber qual será o resultado da ação atômica. O superior adquire responsabilidade de recuperação quando decide por *COMMIT*, pois ele precisa saber se todos os subordinados já colocaram seus dados no estado final após uma falha, para poder apagar informações sobre a ação atômica. Os registros de *log* são armazenados para que informações sobre recuperação estejam disponíveis após a recuperação de falhas. Os dados da ação atômica necessários para recuperação podem ser eliminados somente quando a responsabilidade de recuperação do usuário terminar.

4.4 Comportamento de um Nó Intermediário

As ações de um nó intermediário são a composição de suas ações como superior e subordinado, mas respondendo como subordinado de acordo com o estado da sub-árvore da qual é raiz, e respondendo como superior considerando algumas ações vindas do seu próprio superior.

Deste modo, por exemplo, um nó intermediário só envia uma primitiva C-READY.req a seu superior se já tiver recebido primitivas C-READY.ind de todos os seus subordinados, e só pode enviar uma primitiva C-COMMIT.req a seus subordinados se receber uma primitiva C-COMMIT.ind de seu superior. Em casos de recuperação, também, um nó intermediário pode ter que consultar antes seu superior, para atender a alguma tentativa de recuperação de seus subordinados.

Há duas pequenas particularidades do comportamento de um nó intermediário que não se derivam diretamente da ação conjunta das funções de superior e subordinado comentadas até a seção anterior. A primeira é que, ao enviar uma primitiva C-READY.req a seu superior, o usuário intermediário tem que ter antes armazenado informações para recuperação sobre seus subordinados, para que possa, caso haja um erro, enviar informações a eles. A segunda é que o usuário intermediário pode enviar C-COMMIT.rsp para seu superior mesmo que não tenha recebido C-COMMIT.cnf de seu subordinados. Basta que informações de recuperação para os subordinados dos quais ainda não recebeu esta primitiva sejam armazenadas de forma segura, indicando esta resolução de seu superior. Age também de modo análogo ao receber uma primitiva do tipo C-ROLLBACK.ind, vinda de seu superior, mas neste caso não armazena informações de recuperação. Um nó intermediário pode decidir por *ROLLBACK* em relação a um (ou a alguns) de seus subordinados ou a seu superior a qualquer momento antes de ter enviado uma primitiva C-READY.req a seu superior. No caso de enviar C-ROLLBACK.req a seu superior, o nó intermediário propaga a decisão a seus subordinados. As ações do nó intermediário para outras situações do protocolo são facilmente derivadas.

Em casos em que ocorreram falhas nas situações do parágrafo anterior, ao invés de serem enviadas ou recebidas as primitivas mencionadas, as primitivas do serviço C-RECOVER, com parâmetros adequados, é que seriam usadas. Por exemplo, o subordinado enviaria a seu superior uma primitiva C-RECOVER(ready).req ao invés da primitiva C-READY.req.

4.5 Decisões Heurísticas

A partir do momento em que um subordinado decide oferecer efetivação a seu superior, ele deve aguardar a decisão final da ação atômica, que pode ser *COMMIT* ou *ROLLBACK*. Para um subordinado, o período que vai do oferecimento de efetivação ao recebimento da decisão é chamado de *período de dúvida*, pois ele não sabe o resultado da ação atômica.

Como dito anteriormente, para fins de controle de concorrência, os recursos utilizados por uma ação atômica devem ser protegidos e só liberados no final da ação atômica. Pode acontecer, então, que um subordinado ofereça efetivação, mas que o resultado da ação atômica, por um motivo qualquer, demore a chegar e isto não seja aceitável pela aplicação. Por exemplo, os dados bloqueados (que pode ser feito com o uso de *locks*) podem ser necessários a outras aplicações, que não possam esperar pelo término desta ação atômica. O subordinado pode, com isto, decidir ele mesmo por colocar os dados ligados, ou parte deles, no estado inicial, final ou em um estado intermediário, sem conhecer o resultado da ação atômica. Uma decisão deste tipo é chamada de *Decisão Heurística* e pode levar a inconsistência entre os nós da árvore, caso a decisão heurística tomada não seja igual ao resultado da ação atômica que for decidida pelo mestre.

O CCR não fornece mecanismos para comunicar decisões heurísticas, nem para tratá-las. Tomar decisões heurísticas depende dos próprios usuários e das aplicações e eles mesmos devem-se encarregar de reparar as inconsistências que podem surgir de tais decisões.

4.6 O Protocolo CCR

Nesta seção será apresentado o protocolo do ASE CCR. Será mostrado o seu relacionamento com as camadas inferiores do RM-OSI e o que a máquina de estados realiza ao receber eventos. Deve-se observar que o protocolo CCR é bastante simples. As ações que devem ser feitas para que se mantenha a consistência dos dados ligados, como realizar controle de concorrência e armazenar registros de *log*, devem ser feitas pelos usuários do CCR, e não pela implementação do protocolo. Estas atividades não se referem a aspectos de comunicação e podem variar de aplicação para aplicação. Não faz parte do RM-OSI a padronização de tais procedimentos. A implementação do protocolo vai tratar dos aspectos de geração de APDUs e primitivas, do controle de seqüenciamento dos eventos recebidos (recepção de primitivas e de APDUs), do controle de colisões e do uso de serviços e recursos da camada de Apresentação para a emissão das APDUs.

4.6.1 Procedimentos do protocolo

O CCR assume que uma associação tem que ter sido previamente estabelecida para suportar um ramo de ação atômica. Através desta associação a aplicação enviará os seus dados e os intercalará com APDUs do CCR.

Os eventos que podem ocorrer na máquina de estados do CCR são os que representam recepção de primitivas emitidas por seu usuário ou a chegada de APDUs, vindas do sistema remoto. Todas as APDUs do CCR são enviadas diretamente como valores dos campos de dados do usuário de primitivas da camada de Apresentação.

Para cada primitiva emitida por seu usuário é gerada uma APDU. De modo inverso, para cada APDU recebida, uma primitiva é emitida a seu usuário. Uma primitiva do CCR gera a criação de sempre a mesma APDU e vice-versa. Os mapeamentos são mostrados na tabela 4.1. A primeira

coluna indica a primitiva que é emitida pelo usuário ao CCR, a coluna do meio indica a APDU gerada e a coluna da direita indica a primitiva que será emitida no sistema remoto, na chegada da APDU.

Prim. emitida	*APDU*	*Prim. recebida*
C-BEGIN.req	C-BEGIN-RI	C-BEGIN.ind
C-BEGIN.rsp	C-BEGIN-RC	C-BEGIN.cnf
C-PREPARE.req	C-PREPARE-RI	C-PREPARE.ind
C-READY.req	C-READY-RI	C-READY.ind
C-COMMIT.req	C-COMMIT-RI	C-COMMIT.ind
C-COMMIT.rsp	C-COMMIT-RC	C-COMMIT.cnf
C-ROLLBACK.req	C-ROLLBACK-RI	C-ROLLBACK.ind
C-ROLLBACK.rsp	C-ROLLBACK-RC	C-ROLLBACK.cnf
C-RECOVER.req	C-RECOVER-RI	C-RECOVER.ind
C-RECOVER.rsp	C-RECOVER-RC	C-RECOVER.cnf

Tabela 4.1: Correspondência entre APDUs e primitivas

Com exceção da APDU C-BEGIN-RI, todas as outras possuem campos que correspondem exatamente aos parâmetros das primitivas a partir das quais são geradas e vice-versa. Quando uma APDU for criada, portanto, os valores de seus campos serão os mesmos dos parâmetros correspondentes na primitiva. O inverso ocorrerá quando uma primitiva for ser emitida ao usuário, a partir da recepção de uma APDU. Os campos das APDUs do CCR são mostrados na tabela 4.2, e os seus significados são os mesmos dos parâmetros de mesmo nome das primitivas, apresentados na seção 4.2.

No caso da APDU C-BEGIN-RI, esta também possui correspondentes nos parâmetros das primitivas C-BEGIN.req e C-BEGIN.ind. No entanto estas primitivas possuem, no parâmetro *identificador do ramo da ação atômica*, um componente usado para transmitir o nome do superior, que não possui equivalente em nenhum campo da APDU. Isto ocorre porque, como mencionado na seção 4.2, o nome do superior é o título da AE cuja AEI está solicitando o início do ramo e a especificação do CCR impõe que os títulos de AE de cada uma das AEIs que representem usuários de um ramo sejam informados pelos sistemas comunicantes um ao outro durante o estabelecimento da associação que for suportar este ramo. Estas informações irão em APDUs do protocolo ACSE e não precisam, portanto, serem enviadas novamente. Como mencionado na seção 4.2 os títulos das AEs serão usados para identificar os ramos da ação atômica.

Uma APDU do CCR pode ser enviada isoladamente ou concatenada com outras APDUs do próprio CCR ou de outro ASE. O documento [ISO9805] especifica quais concatenações de APDUs são permitidas e em quais primitivas da camada de Apresentação as APDUs devem ser enviadas (como valores de seus parâmetros de dados do usuário). Quando não há concatenação, ou a concatenação for as das APDUs C-COMMIT-RI + C-BEGIN-RI e C-ROLLBACK-RI + C-BEGIN-RI, concatenações estas causadas pela emissão conjunta das primitivas comentadas na seção 4.2, as primitivas da camada de Apresentação em cujos parâmetros de dados de usuário elas são enviadas estão na tabela 4.3. A primeira coluna desta tabela contém primitivas do CCR. A segunda coluna contém as APDUs geradas pelo CCR pela recepção destas primitivas e a terceira coluna apresenta

APDU	*Campos*
C-BEGIN-RI	• ident. da ação atômica: – nome do mestre – sufixo • sufixo do ident. do ramo da ação atômica • dados do usuário
C-BEGIN-RC C-PREPARE-RI C-READY-RI C-COMMIT-RI C-COMMIT-RC C-ROLLBACK-RI C-ROLLBACK-RC	dados do usuario
C-RECOVER-RI C-RECOVER-RC	• ident. da ação atômica: – nome do mestre – sufixo • ident. do ramo da ação atômica: – nome do superior – sufixo • estado de recuperação • dados do usuário

Tabela 4.2: APDUs do CCR e seus campos

as primitivas da camada de Apresentação. As primitivas unidas pelo sinal "+" na primeira coluna indicam que o usuário do CCR deve emiti-las juntas, gerando a concatenação das APDUs formadas a partir delas. Na terceira coluna, quando aparece o termo *restart* escrito entre parênteses, significa que este deve ser o valor do parâmetro *tipo de ressincronização* das primitivas do serviço P-RESYNCHRONIZE [CCITT216], do mesmo modo como usado anteriormente.

Quais são as seqüências de concatenações permitidas por [ISO9805] e os seus mapeamentos em primitivas da camada de Apresentação não serão tratados aqui, pois no SISDI-OSI só serão feitas as concatenações que aparecem na tabela 4.3. Quando as APDUs do CCR forem concatenadas com APDUs de outros protocolos, regras adicionais devem ser acrescentadas no contexto de aplicação.

Todas as primitivas mostradas na tabela 4.3 existem na camada de Apresentação para que entidades da camada de Aplicação possam acessar serviços correspondentes da camada de Sessão (no modelo RM-OSI entidades de uma camada só se comunicam com entidades de camadas adjacentes). Os serviços P-SYNC-MAJOR, P-SYNC-MINOR, P-TYPED-DATA e P-RESYNCHRONIZE da camada de Apresentação correspondem, respectivamente, aos serviços S-SYNC-MAJOR, S-SYNC-MINOR, S-TYPED-DATA e S-RESYNCHRONIZE da camada de Sessão. Por serem serviços correspondentes, há correspondência também entre as primitivas dos serviços.

O serviço S-SYNC-MAJOR e o serviço S-SYNC-MINOR são usados para se estabelecerem, respectivamente, pontos de sincronização maiores e menores no diálogo através de uma associação[1]. Na seção 2.3.4 há uma explicação do uso destes pontos. O serviço S-RESYNCHRONIZE é utilizado para as entidades usuárias comunicantes restabelecerem a comunicação a um destes pontos definidos no diálogo. O serviço S-TYPED-DATA é utilizado para o envio de dados. Além deste serviço a camada de Sessão oferece mais três para a transferência de dados, cada um com características próprias, cujas diferenças não são importantes para este texto. Basta salientar que o serviço S-TYPED-DATA não está sujeito a restrições de posse de *tokens* da camada de Sessão.

Observe a relação semântica que há entre o efeito do uso dos serviços do CCR em relação à ação atômica e o efeito causado pelas primitivas da camada de Sessão na conexão que suporta a associação: o ponto de sincronização associado ao serviço C-BEGIN corresponde a um ponto em que a ação atômica poderá retornar (caso haja *ROLLBACK*); o ponto de sincronização maior impede que a conexão de Sessão retorne a um ponto anterior, de maneira análoga a uma decisão por *COMMIT* impedir que a ação atômica retorne a um estado anterior; o serviço S-RESYNCHRONIZE é usado para que os seus usuários se ressincronizem em um ponto da conexão, de modo análogo ao serviço C-ROLLBACK causar o retorno ao estado inicial da ação atômica. Os serviços C-PREPARE e C-READY simplesmente passam informações de um usuário a outro e, portanto, são mapeados em um serviço de transferência de dados, o P-TYPED-DATA. O serviço C-RECOVER possui uma semântica de ressincronização, mas é mapeado no serviço P-TYPED-DATA pois a recuperação é feita utilizando-se uma associação diferente da que havia suportado o ramo antes da falha, o que faz com que os pontos de sincronização não estejam mais definidos.

A camada de Sessão impõe como restrição para a invocação de uma primitiva S-SYNC-MINOR.req, a posse do *token* de sincronização menor [ISO8326]. Por isto, para que o usuário do CCR invoque a primitiva C-BEGIN.req, ele deve estar de posse deste *token*, pois a invocação desta primitiva gera a criação da APDU C-BEGIN-RI, que é enviada utilizando-se a primitiva

[1]Precisamente, os pontos de sincronização estão associados à conexão de Sessão que suporta a associação. Neste texto, no entanto, serão referenciados como relacionados a uma associação, o que não fica incorreto, pois há um relacionamento de um para um entre associações e conexões de Sessão. Este uso deixa mais breves algumas referências feitas a estes pontos neste texto.

Primitiva do CCR	*APDU do CCR*	*Primitiva da Apresentação*
C-BEGIN.req/ind	C-BEGIN-RI	P-SYNC-MINOR.req/ind
C-BEGIN.rsp/cnf	C-BEGIN-RC	P-SYNC-MINOR.rsp/cnf
C-BEGIN.rsp/cnf quando C-BEGIN.req foi emitido com C-ROLLBACK.req ou C-COMMIT.req	C-BEGIN-RC	P-TYPED-DATA.req/ind
C-PREPARE.req/ind	C-PREPARE-RI	P-TYPED-DATA.req/ind
C-READY.req/ind	C-READY-RI	P-TYPED-DATA.req/ind
C-ROLLBACK.req/ind	C-ROLLBACK-RI	P-RESYNC(restart).req/ind
C-ROLLBACK.rsp/cnf	C-ROLLBACK-RC	P-RESYNC(restart).rsp/cnf
C-ROLLBACK.req/ind + C-BEGIN.req/ind	C-ROLLBACK-RI seguida de C-BEGIN-RI	P-RESYNC(restart).req/ind
C-ROLLBACK.rsp/cnf + C-BEGIN.rsp/cnf	C-ROLLBACK-RC seguida de C-BEGIN-RC	P-RESYNC(restart).rsp/cnf
Colisão ilustrada na figura 4.6	C-ROLLBACK-RC seguida de C-BEGIN-RI	P-RESYNC(restart).rsp/cnf
C-COMMIT.req/ind	C-COMMIT-RI	P-SYNC-MAJOR.req/ind
C-COMMIT.rsp/cnf	C-COMMIT-RC	P-SYNC-MAJOR.rsp/cnf
C-COMMIT.req/ind + C-BEGIN.req/ind	C-COMMIT-RI seguida de C-BEGIN-RI	P-SYNC-MAJOR.req/ind
C-COMMIT.rsp/cnf + C-BEGIN.rsp/cnf	C-COMMIT-RC seguida de C-BEGIN-RC	P-SYNC-MAJOR.rsp/cnf
C-RECOVER.req/ind	C-RECOVER-RI	P-TYPED-DATA.req/ind
C-RECOVER.rsp/cnf	C-RECOVER-RC	P-TYPED-DATA.req/ind

Tabela 4.3: Mapeamento de APDUs do CCR e primitivas

P-SYNC-MINOR.req. De modo análogo, para se invocar a primitiva C-COMMIT.req, o usuário deve estar de posse dos *tokens* de sincronização maior/atividade e do de sincronização menor, pois a invocação desta primitiva gerará a criação da APDU C-COMMIT-RI, que é enviada na primitiva P-SYNC-MAJOR.req, e a camada de Sessão impõe como restrição para a invocação da primitiva S-SYNC-MAJOR.req a posse destes *tokens*. Nos documentos [ISO9805] e [ISO9804] não se exige a posse do *token* de sincronização menor para este caso, mas isto constitui-se uma falha, que pode ser confirmada por [ISO8326]. O mesmo acontece com a emissão conjunta das primitivas C-COMMIT.req e C-BEGIN.req, que também deve exigir a posse dos dois *tokens*, e em [ISO9805] e [ISO9804] só se exige a posse do de sincronização maior/atividade.

A emissão das primitivas S-SYNC-MAJOR e S-SYNC-MINOR também requerem a posse do *token* de dados, quando este estiver sendo usado [ISO8326]. Nesta condição então a posse deste *token* será também necessária para a emissão das primitivas C-COMMIT.req e C-BEGIN.req. Quando a primitiva C-BEGIN.req for emitida juntamente com a primitiva C-ROLLBACK.req, a posse de *token* não é necessária, uma vez que o mapeamento das APDUs geradas será feita no campo de dados de usuário da primitiva P-RESYNCHRONIZE.req, que não exige *tokens*. O CCR assume que é responsabilidade de seu usuário observar as exigências sobre a posse de *tokens* para a emissão de seus serviços.

Os valores dos parâmetros das primitivas da camada de Apresentação que devem ser usados quando estas primitivas forem usadas para a emissão das APDUs do CCR, como mostrado na tabela 4.3, são preenchidos da seguinte maneira:

1. **P-SYNC-MINOR.req/ind**

   tipo é atribuído o valor *opcional*, o que indica que, pela camada de Sessão, a confirmação desta primitiva não é obrigatória. O CCR, no entanto, impõe regras para a obrigatoriedade ou não da emissão da confirmação, como será mostrado mais à frente nesta seção;

   número do ponto serial de sincronização este valor corresponde ao próximo valor seqüencial para os números dos pontos de sincronização da associação. O usuário do CCR (incluindo AEI) deve manter informações sobre o estado da associação neste ponto, para o caso de haver um *ROLLBACK* e a associação ter que retornar a este estado;

   dados do usuário contém a APDU C-BEGIN-RI.

2. **P-SYNC-MINOR.rsp/cnf**

   número do ponto serial de sincronização possui o mesmo valor do parâmetro de mesmo nome da primitiva P-SYNC-MINOR.ind correspondente;

   dados do usuário contém a APDU C-BEGIN-RC.

3. **P-TYPED-DATA.req/ind**

   dados do usuário contém a APDU a ser enviada. As APDUs que podem ser transmitidas como valores deste parâmetro são indicadas na tabela 4.3.

4. **P-SYNC-MAJOR.req/ind**

número do ponto serial de sincronização possui, de modo análogo ao parâmetro de mesmo nome das primitivas P-SYNC-MINOR.req/ind, o próximo número seqüencial dos pontos de sincronização para a associação;

dados do usuário contém a APDU ou APDUs associadas a estas primitivas na tabela 4.3.

5. **P-SYNC-MAJOR.rsp/cnf**

   dados do usuário é usada para transportar APDUs associadas a estas primitivas na tabela 4.3.

6. **P-RESYNCHRONIZE.req/ind**

   tipo de ressincronização recebe sempre o valor *restart*. Este parâmetro indica que a ressincronização deve ser para um ponto de sincronização anterior no diálogo, que não seja um ponto anterior ao último ponto de sincronização maior confirmado. Esta primitiva possui outros valores para este parâmetro, mas que não são usados para o envio das APDUs do CCR. Como o serviço P-RESYNCHRONIZE existe apenas para que entidades da camada de Aplicação acessem serviços da camada de Sessão, maiores informações sobre esta primitiva são encontradas em [ISO8326], na descrição do serviço S-RESYNCHRONIZE;

   número do ponto serial de sincronização recebe o valor do parâmetro de mesmo nome da primitiva que transportou a APDU C-BEGIN-RI. O documento [ISO9805] especifica que o valor deste parâmetro teria que ser o número do menor ponto de sincronização para o qual a primitiva P-RESYNCHRONIZE.req poderia ser usada com o parâmetro *restart*, caso este número seja maior que o do ponto de sincronização associado ao C-BEGIN-RI. No entanto, pela análise do uso dos pontos de sincronização em [ISO8326], este caso nunca acontece (pelo menos quando se faz o *uso direto do CCR*, como será explicado na seção 4.7, que é o definido no documento do protocolo do CCR);

   tokens se quem emite esta primitiva é um subordinado, este campo deve conter todas as tokens que estiverem sendo usadas.

   dados do usuário usado para enviar as APDUs associadas a estas primitivas na tabela 4.3.

7. **P-RESYNCHRONIZE.rsp/cnf**

   número do ponto serial de sincronização possui o mesmo valor do parâmetro de mesmo nome da primitiva P-RESYNCHRONIZE.ind correspondente;

   dados do usuário usado para transmitir as APDUs associadas a estas primitivas na tabela 4.3.

O protocolo CCR, como dito na seção 4.1, possui uma pequena variação do protocolo *Presumed Rollback*. Esta variação é o fato de enquanto no CCR o serviço C-BEGIN ser opcionalmente confirmado e o serviço C-ROLLBACK ser sempre confirmado, no *Presumed Rollback* estes serviços seriam sempre não confirmados. No CCR isto acontece devido a algumas particularidades das camadas inferiores do RM-OSI, como explicado a seguir.

O fato do C-ROLLBACK ser sempre confirmado vem do fato de as primitivas deste serviço gerarem APDUs que são enviadas utilizando-se o serviço P-RESYNCHRONIZE da camada de Apresentação, que é um serviço obrigatoriamente confirmado. Este serviço só permite o uso de alguns outros poucos até que venha a sua confirmação. O fato de o serviço C-BEGIN ser opcionalmente confirmado depende do uso ou não do serviço TRANSPORT EXPEDITED, da camada de Transporte, pela camada de Sessão. Se este serviço for usado, o protocolo impõe que:

- o serviço C-BEGIN deve ser confirmado; e
- após um usuário ter emitido uma primitiva do tipo C-BEGIN.req, ele não pode enviar uma primitiva do tipo C-ROLLBACK.req até o recebimento da confirmação, ou seja, C-BEGIN.cnf.

Se o serviço TRANSPORT EXPEDITED não for usado, estas restrições não se aplicam e o serviço pode ser não confirmado.

Esta restrição existe por causa de mapeamentos de PDUs da camada de Sessão em serviços da camada de Transporte. Esta camada fornece dois tipos de transferência de dados [Bla89]: o *normal*, sujeito a controle de fluxo durante a transmissão, e o *expedited*, que não está sujeito a este controle, fazendo com que tenha prioridade na transmissão sobre o primeiro tipo. Quando este serviço é fornecido na camada de Transporte, pelas regras de mapeamentos de PDUs da camada de Sessão em primitivas de serviços da camada de Transporte, ele será usado na transferência da APDU C-ROLLBACK-RI, enquanto a APDU C-BEGIN-RI será transferida através do fluxo NORMAL de dados [ISO8327]. Se as restrições acima não existissem, poderia ocorrer a situação de um superior emitir a primitiva C-BEGIN.req e logo depois a primitiva C-ROLLBACK.req, mas a APDU C-ROLLBACK-RI chegar primeiro do que a APDU C-BEGIN-RI no sistema remoto, devido ao uso do serviço *expedited*. O serviço S-RESYNCHRONIZE, da camada de Sessão, que é usado para transmitir a APDU C-ROLLBACK-RI, causa a eliminação das APDUs que chegam até a sua confirmação ser feita. Isto causaria a perda da APDU C-BEGIN-RI, o que faz com que o início da ação atômica fique indeterminado para o subordinado.

O CCR exige que as seguintes unidades funcionais sejam usadas na associação: *kernel*, *typed data*, *major synchronize*, *minor synchronize* e *resynchronize* da camada de Sessão, a unidade *kernel* da camada de Apresentação e as outras desta camada que existem para repassar os serviços destas unidades da Sessão. Estas unidades são as que contêm as primitivas da Apresentação usadas para transmitir as APDUs do CCR e as que possuem as primitivas da camada de Sessão correspondentes.

4.6.2 Colisões

No CCR as colisões são resolvidas de acordo com as regras para colisão das APDUs da camada de Sessão nas quais as APDUs do CCR são transportadas. Se uma das APDUs for eliminada em conseqüência de uma colisão, somente um dos usuários do ramo receberá uma primitiva (do tipo *indication* ou *confirm*, resultante da chegada da APDU). Caso nenhuma das APDUs seja eliminada, as duas primitivas resultantes são emitidas, uma em cada sistema. Em alguns casos em que as APDUs não são eliminadas pode-se aparentar uma quebra na seqüência normal de recepção das primitivas do CCR, mas é previsto no protocolo. Um exemplo disto é a colisão entre as APDUs C-READY-RI e C-PREPARE-RI, quando o sistema emissor da APDU C-READY-RI, mesmo já tendo enviado a resposta de que pode realizar as tarefas (a APDU C-READY-RI), vai receber a APDU C-PREPARE-RI, que justamente requisita o envio da resposta sobre a possibilidade de

realização das tarefas. A primitiva C-PREPARE.ind é emitida ao usuário normalmente. Colisões com C-ROLLBACK-RI fazem com que uma das APDUs seja descartada (colisão de APDUs geradas por primitivas do serviço S-RESYNCHRONIZE).

O único cuidado que teve que ser tomado na implementação do protocolo CCR em relação a colisões foi para o tratamento da colisão das APDUs concatenadas C-ROLLBACK-RI e C-BEGIN-RI, enviadas por um superior, com uma APDU C-ROLLBACK-RI, enviada por um subordinado. Estas APDUs são enviadas em primitivas S-RESYNCHRONIZE, cuja regra de colisão de suas PDUs pode fazer com que as APDUs concatenadas sejam descartadas[2]. Neste caso o superior, ao enviar a APDU C-ROLLBACK-RC, deve enviá-la concatenada com uma cópia da APDU C-BEGIN-RI descartada. A figura 4.6 ilustra a situação. Observe que nesta situação e em outras com a APDU C-ROLLBACK-RI há perda de APDUs, o que faz com que a entrega de dados de usuário emitidos nestas APDUs não seja garantida.

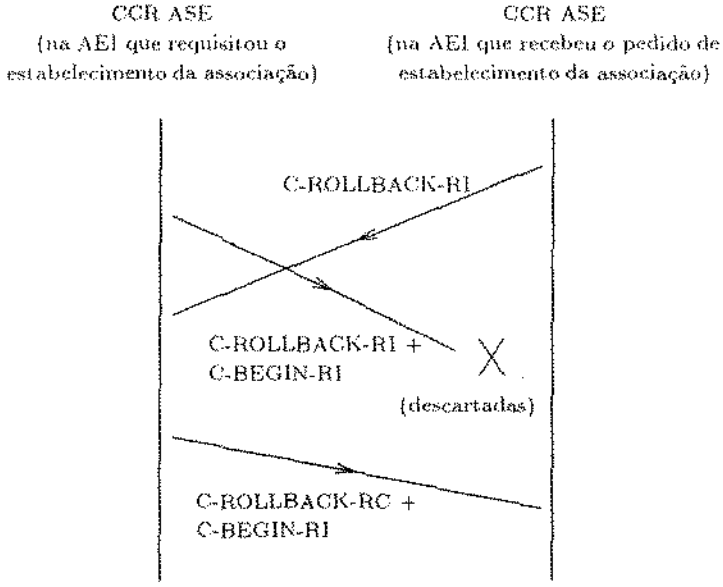

Figura 4.6: Caso de colisão de C-ROLLBACK-RI, com reenvio de C-BEGIN-RI

Além disto a definição de serviços do CCR ([ISO9804]) exige a posse do *token* de sincronização menor para a emissão da primitiva C-RECOVER.req. Isto é feito para se evitar a colisão entre as APDUs C-RECOVER-RI e C-BEGIN-RI (a emissão da primitiva C-BEGIN.req requer a posse deste *token*). Desta maneira, sobre uma associação, ocorre ou o estabelecimento de um novo ramo de ação atômica ou uma recuperação. Esta exigência de posse de *token* evita também a colisão entre duas APDUs C-RECOVER-RI. Com isto, se ambos os usuários quiserem fazer uma recuperação sobre a associação, somente um deles poderá enviar a APDU C-RECOVER-RI, ou seja, invocar uma primitiva C-RECOVER.req, mesmo que ambos tenham responsabilidade de recuperação. Quando

[2]Segundo as regras da camada de Sessão, quando houver uma colisão entre os serviços S-RESYNCHRONIZE.req, ambas com o mesmo número de ponto de sincronização, que é o caso quando são usadas para transmitir as APDUs do CCR, ganha o sistema que requisitou o estabelecimento da conexão na qual a colisão ocorreu. A APDU do sistema perdedor é eliminada.

um superior, no entanto, for emitir uma primitiva C-RECOVER.req como resposta à recepção de uma primitiva C-RECOVER.ind (ver seção 4.3), a posse deste *token* não é obrigatória, pois não poderá ocorrer estas colisões neste caso.

4.7 Uso do CCR por um Serviço Principal Cooperativo

O uso dos serviços do CCR e o protocolo descritos neste capítulo é chamado de *uso direto do CCR*. Pode ser o caso de o CCR não poder ser utilizado desta maneira, por exemplo, pelo fato de o seu usuário utilizar recursos da camada de Apresentação de modo incompatível com o uso deles pelo CCR. Por exemplo, o usuário querer que duas atividades da camada de Sessão (ver seção 2.3.4) sejam realizadas como uma única ação atômica. Quando atividades são usadas em uma associação os serviços da camada de Sessão usados pelo CCR só podem ser usados dentro de uma atividade, o que torna a aplicação impossível.

Neste caso toda a semântica do CCR deve ser incorporada em primitivas de serviços e em APDUs de seu usuário. Este uso do CCR é chamado de *uso por um Serviço Principal Cooperativo*. Neste caso a definição dos serviços, das regras de seqüenciamento e formato das APDUs do CCR continuam válidos, mas devem-se definir novas regras para envio das APDUs, que podem passar a fazer parte de APDUs do próprio usuário; em que primitivas da camada de Apresentação as APDUs irão; como deve ser o uso dos parâmetros destas primitivas; quais concatenações de APDUs podem ser feitas e como estas concatenações devem ser enviadas, dentre outras regras, de modo a que os serviços e o protocolo se comportem semanticamente de maneira equivalente ao do *uso direto do CCR*.

Capítulo 5

A Implementação do CCR e de Componentes Auxiliares

Este capítulo descreve como foram implementados a especificação do protocolo CCR (documentos [ISO9804] e [ISO9805]) e alguns componentes auxiliares, necessários para que a implementação do protocolo pudesse ser testada. Esta implementação foi feita de acordo com as definições de funcionalidades para os componentes da camada de Aplicação mostradas no capítulo 2 e, como faz parte do SISDI-OSI, segue a estrutura geral deste sistema, mostrada no capítulo 3.

Na seção 5.1 mostram-se os componentes que fazem parte da implementação e as interações que há entre eles. Na seção 5.2 comenta-se sobre o uso de ESTELLE e sobre o ambiente EWS, usado para a implementação. Na seção 5.3 descreve-se a especificação feita em ESTELLE para a geração do código em C para o protocolo CCR. Na seção 5.4 comenta-se sobre os principais aspectos da implementação do SACF que suporta a implementação deste protocolo. Finalmente, na seção 5.5, é mostrado como funciona um simulador das camadas inferiores do modelo RM-OSI e da rede de comunicação, feito para simular a comunicação entre dois sistemas distintos.

As descrições das implementações neste capítulo assumem do leitor um conhecimento de ESTELLE. No anexo A, no entanto, está descrito o subconjunto desta linguagem que foi usado nas implementações.

5.1 Os Componentes Implementados e as Interações entre eles

A implementação feita consiste dos componentes necessários ao estabelecimento de uma associação, para suportar um ramo de ação atômica, e à transmissão das APDUs do CCR, para a execução propriamente dita do protocolo. A figura 5.1 apresenta um esquema dos componentes implementados e das interações que há entre eles.

Nesta figura estão representados os ASEs CCR e ACSE, um *Simulador das Camadas Inferiores*, um componente SACF e um componente chamado *USUÁRIO*, dividido internamente em três outros componentes. O *Simulador das Camadas Inferiores* simula a existência das seis camadas inferiores do modelo RM-OSI (da Apresentação à Física) e de uma rede de comunicação para a troca de dados entre dois sistemas distintos. Este simulador interage com o ACSE e o SACF através de primitivas de serviço da camada de Apresentação, que é a camada mais alta do modelo, dentre as simuladas.

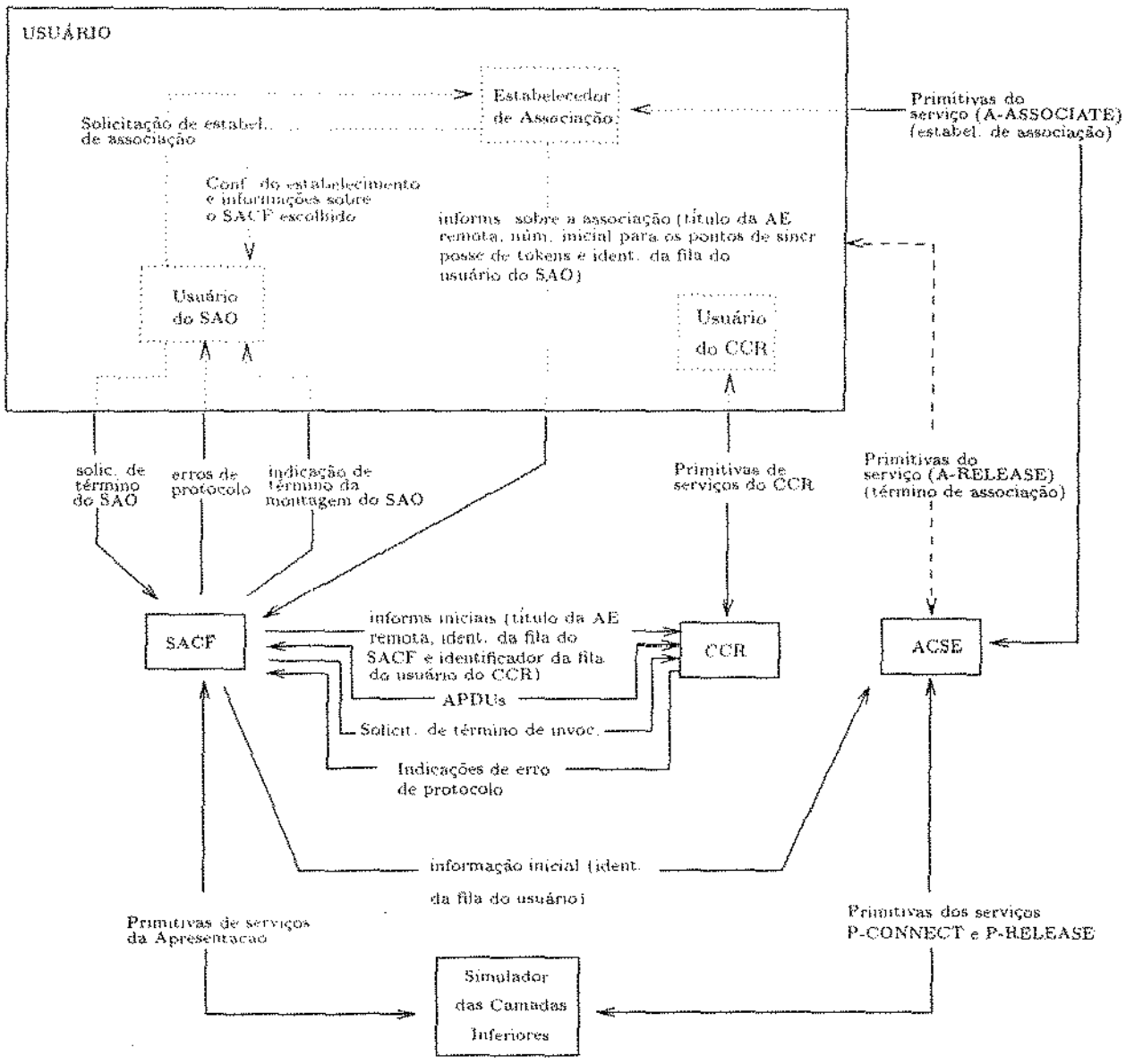

Figura 5.1: Componentes implementados e seus relacionamentos

O ACSE já havia sido implementado no SISDI-MAP [Pag91] e sua implementação foi apenas adaptada ao SISDI-OSI. A simulação das camadas inferiores foi feita, pois as implementações destas camadas não foram realizadas ou as partes implementadas de algumas delas não foram ainda convertidas para o SISDI-OSI. O *Simulador das Camadas Inferiores* simula uma comunicação entre dois sistemas distintos através do mapeamento de conexões de apresentação e do uso de duas invocações de componentes da camada de Aplicação. A sua implementação é discutida na seção 5.5.

O componente *SACF* representa as funções atribuídas à parte específica de contexto de aplicação de SACFs (ver seção 2.4.4) de acordo com a definição de funcionalidades para a camada de Aplicação feita no capítulo 2, mas apenas aquelas relacionadas ao *uso direto do CCR*. O *uso direto do CCR* é uma das maneiras de se utilizar os serviços e transmitir as APDUs do CCR, como comentado no capítulo 4.

De acordo com este modelo de funcionalidades, procura-se implementar os ASEs, no caso, o CCR, de modo a que realizem apenas as funções específicas do protocolo, independentes do contexto de aplicação usado. O componente SACF implementado recebe as APDUs do CCR, concatena-as se for o caso e as mapeia em parâmetros de dados do usuário de primitivas da camada de Apresentação, para serem enviadas ao sistema remoto. Na recepção de primitivas da camada de Apresentação, o SACF extrai as APDUs e as envia ao CCR (por enquanto, só está tratando APDUs do CCR, interagindo, portanto, apenas com este ASE). O SACF faz ainda o controle sobre os números de pontos de sincronização a serem usados nos parâmetros das primitivas da camada de Apresentação. O CCR e o SACF foram implementados em ESTELLE. Todos os outros componentes, em C.

O componente *USUÁRIO* aparece na figura dividido internamente em três componentes: o *ESTABELECEDOR DE ASSOCIAÇÃO*, o *USUÁRIO DO CCR* e o *USUÁRIO DO SAO*. O componente *ESTABELECEDOR DE ASSOCIAÇÃO* representa o componente de mesmo nome descrito na seção 2.4.4, que é o responsável pelo estabelecimento da associação e seleção do SACF adequado, de acordo com o contexto de aplicação negociado. O *USUÁRIO DO CCR* representa o componente que for emitir primitivas de serviço do CCR de acordo com o contexto de aplicação. Este componente pode ser um outro ASE, o próprio SACF, o MACF ou o AP. O *USUÁRIO DO SAO* corresponde ao componente (MACF ou AP) que interagir com o SACF para o uso da associação. O componente *USUÁRIO* representa estes três componentes. Na implementação, no entanto, como será visto, eles não existirão. Esta decomposição foi mostrada na figura apenas para se entender o relacionamento que há entre alguns componentes da camada de Aplicação relacionados com esta implementação, segundo o modelo de funcionalidades.

As implementações do CCR, do SACF e do ACSE seguem exatamente o modelo geral de implementação para o SISDI-OSI, descrito no capítulo 3. Cada um destes componentes está, portanto, implementado como um processo UNIX independente, que simula internamente as suas várias instâncias de uso, possui uma única fila para recepção de mensagens e acessa uma área de memória compartilhada, para o armazenamento do conteúdo das mensagens. As APDUs e primitivas são transmitidas entre os componentes como comentado naquele capítulo.

As simulações internas das instâncias do CCR e SACF foram feitas utilizando-se recursos próprios de ESTELLE (instâncias de módulos). A simulação das instâncias do ACSE foi feita utilizando-se tabelas que armazenam o estado em que cada instância está. Na recepção de um evento (primitiva ou APDU), este é tratado de acordo com o estado em que a instância em que está associado se encontra.

O *Simulador das Camadas Inferiores* também segue o modelo geral, mas não possui instâncias

internas a serem simuladas. O componente *USUÁRIO* foi implementado como dois processos, um para o envio de mensagens e outro para recepção, possuindo, portanto, uma fila associada a apenas este último processo. Não possui também instâncias internas a serem simuladas.

Por ser a união dos componentes citados anteriormente, o *USUÁRIO* controla o estabelecimento de associações para a troca das APDUs do CCR, invoca as primitivas de serviço deste protocolo, envia mensagens ao SACF e trata respostas vindas destes componentes. Além disto, controla também a liberação da associação. A sua implementação é bastante simples, apenas para teste da implementação do CCR e do SACF. O seu processo UNIX para envio fornece ao usuário da máquina na qual estas implementações estiverem rodando um *menu* com opções de primitivas a serem invocadas. Estas primitivas podem ser de serviços do ACSE, do CCR ou de controle. O usuário escolhe uma delas e preenche seus parâmetros, caso tenham. A primitiva então é enviada ao ASE adequado, se for uma primitiva de serviço, ou ao SACF, caso seja uma primitiva de controle. Após a emissão o processo passa a esperar uma nova solicitação do usuário da máquina.

A emissão de uma primitiva de serviço do ACSE ou CCR provocaria, na comunicação entre dois sistemas reais e quando não há erros, a emissão de uma primitiva no sistema remoto. Pelo fato de a comunicação entre sistemas distintos ser apenas simulada, a emissão da primitiva feita hipoteticamente no sistema remoto será feita pelo próprio ASE fornecedor do serviço, mas em uma outra de suas invocações (figura 5.2). Esta primitiva será enviada ao processo do *USUÁRIO* que trata de recepção. Este processo imprime na tela os valores dos parâmetros da primitiva.

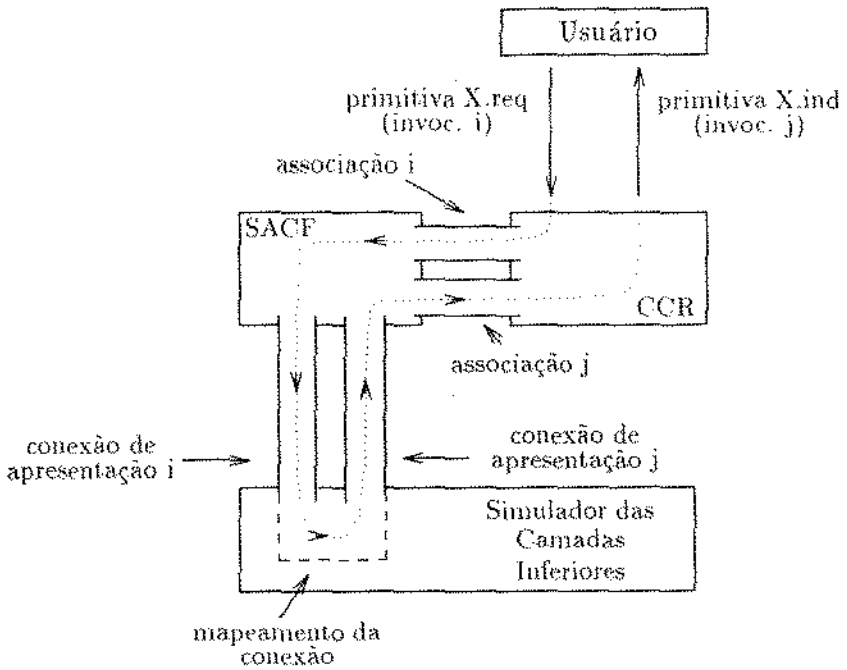

Figura 5.2: Mapeamento de conexões de apresentação

As primitivas de controle fornecidas ao usuário da máquina não geram emissão de primitivas no sistema remoto. Há uma primitiva de controle para o envio das informações iniciais sobre a associação estabelecida e para o término de invocações do SACF. O *USUÁRIO*, através de seu

processo de recepção de mensagem, recebe e imprime também as mensagens de erros detectados pelo CCR. Estes erros são comunicados ao SACF que os passa ao USUÁRIO e podem ocorrer na emissão de uma primitiva ou na recepção de uma APDU.

Deve-se observar que o usuário da máquina controla ambas as associações, nos dois sistemas hipoteticamente comunicantes, e interage com os componentes do sistema através de primitivas para simular a existência dos três componentes internos do USUÁRIO. As interações entre os componentes, comentadas a seguir, portanto, ocorrem nos dois sistemas hipotéticos.

Interações entre os Componentes

As interações entre componentes internos do *USUÁRIO* não ocorrerão, visto como este componente é implementado. No entanto elas foram mostradas tracejadas na figura 5.1 e incluídas na descrição dos relacionamentos que se segue, para esclarecer como conceitualmente os componentes se interagiriam. Com o desenvolvimento do SISDI-OSI, os componentes internos do *USUÁRIO* passarão a ser autônomos.

Como mostrado na figura 5.1, o *USUÁRIO DO SAO* requisita o estabelecimento de associação ao *ESTABELECEDOR DE ASSOCIAÇÃO*. Este controla o estabelecimento, interagindo com o *ACSE* através de primitivas deste ASE (primitivas do serviço A-ASSOCIATE [CCITT217]). O ACSE, por sua vez, interage com a camada de Apresentação, trocando primitivas do serviço P-CONNECT [CCITT216] desta camada, para estabelecer uma conexão de apresentação para suportar a associação. Uma vez estabelecida a associação, o *ESTABELECEDOR DE ASSOCIAÇÃO* selecionaria (nos dois sistemas que se comunicam hipoteticamente) o SACF adequado, de acordo com o contexto de aplicação negociado. Em seguida informaria ao *USUÁRIO DO SAO* a confirmação do estabelecimento da associação, juntamente com informações sobre o SACF escolhido (qual é o contexto de aplicação que representa e seu identificador de fila) e sobre a associação estabelecida (por exemplo, o endereço de apresentação de resposta da AE remota (*responding presentation address*) [CCITT217]. A seguir transfere ao SACF informações sobre o estabelecimento. Na implementação não há ainda negociação de contextos de aplicação. Há também um só SACF. As informações que são atualmente transferidas do *ESTABELECEDOR DE ASSOCIAÇÃO* para o SACF são o título da AE remota (título de AE da AEI com a qual a associação foi estabelecida), o número inicial para os pontos de sincronização, a atribuição inicial de *tokens* e o identificador de fila do processo usuário do SAO. Este identificador é, por enquanto, sempre o da fila associada ao processo de recepção de mensagens do componente *USUÁRIO*. Deve-se observar que a interação comentada aqui do *ESTABELECEDOR DE ASSOCIAÇÃO* com o *USUÁRIO DO SAO* é apenas um exemplo. A interação exata depende de quais funcionalidades se atribuirá ao *ESTABELECEDOR DE ASSOCIAÇÃO*.

O SACF, a seguir, monta o SAO, segundo o contexto de aplicação a ser usado, passando informações iniciais que forem necessárias a cada ASE que fizer parte deste SAO. Ao ACSE será informado qual é o identificador de fila do usuário dos seus serviços A_RELEASE, A_ABORT e A_P_ABORT [CCITT217]. Ao CCR o SACF informa qual é o título da AE remota, fornecido ao SACF pelo *ESTABELECEDOR DE ASSOCIAÇÃO*, e dois identificadores de fila. O título da AE é usado para a emissão da primitiva C-BEGIN.ind, pois esta informação, como comentado na seção 4.6.1, não é transmitida juntamente com a APDU C-BEGIN-RI, que causa a emissão desta primitiva. Os identificadores de fila que o SACF fornece são o identificador de sua própria fila e o da fila do processo que seria o receptor de primitivas de serviço do CCR, segundo o contexto de aplicação. Por enquanto o processo *USUÁRIO* está desempenhando a função de processo receptor

das primitivas do CCR e do ACSE, sendo portanto o seu identificador de fila que está sendo fornecido. Com estes identificadores os ASEs (CCR e ACSE) ficam sabendo para quem enviar as mensagens em cada uma de suas instâncias de uso.

A seguir o SACF envia uma mensagem ao *USUÁRIO DO SAO* informando que o SAO foi montado e que está pronto para receber solicitações de serviços.

Após o estabelecimento da associação, durante a fase de transmissão de dados (execução do protocolo de duas fases), o componente *USUÁRIO DO CCR* passa a interagir com o *CCR*, solicitando serviços através de suas primitivas. O *CCR* interage com o *SACF*, enviando a ele APDUs a serem enviadas na rede e recebendo dele APDUs enviadas pelo sistema remoto. Trocam ainda mensagens de controle, emitidas pelo SACF para requisitar o término de uma invocação do CCR e por este para indicar detecção de erros de protocolo.

O SACF interage com o seu usuário com mensagens análogas, recebendo deste solicitações para término de invocação, que corresponde ao término do SAO, e enviando a ele indicações de erros de protocolos detectados pelos ASEs (por enquanto, só o CCR). O SACF interage com a camada de Apresentação, através de primitivas desta camada, apenas para o envio e recepção de APDUs, por enquanto.

A liberação da associação será controlada pelo USUÁRIO, através do uso das primitivas do serviço A-RELEASE do ACSE [CCITT217]. A linha que une este componente com o ACSE, para indicar a interação para este procedimento, aparece tracejada e não está conectada a nenhum módulo interno do USUÁRIO, pois o componente exato que controla a liberação da associação varia dependendo do contexto de aplicação. O USUÁRIO também vai exercer a função deste componente.

No SISDI-OSI, quando um componente emite uma mensagem a um outro componente, ele deve indicar a qual instância do componente destino a mensagem se refere. As instâncias internas de componentes do sistema são identificadas no SISDI-OSI através de valores inteiros, os *identificadores de invocação*. Estes identificadores são enviados nas mensagens colocadas nas filas dos processos, como indicado na seção 3.5.1. Todas as mensagens recebidas por um componente do SISDI-OSI com um mesmo identificador de invocação são tratadas por ele como se referindo a uma mesma instância sua. Isto serve para associar eventos ocorridos em uma mesma associação. Os relacionamentos descritos anteriormente ocorrem para cada associação de modo independente e podem ocorrer em paralelo. Estes identificadores de invocação permitem separar os eventos de cada associação.

5.2 O Uso de ESTELLE e o Ambiente EWS

A escolha pela linguagem ESTELLE para implementar o CCR e o SACF foi em função de uma outra tese [Fuj] ser a implementação do protocolo TP, a partir de sua especificação nesta linguagem, fornecida em um de seus documentos de especificação [ISO10026-3]. Uma especificação do CCR e do SACF também nesta linguagem permitiria o estudo das interações entre o TP e estes componentes utilizando-se ferramentas simuladoras que atuam sobre o código em ESTELLE. Uma destas ferramentas poderia ser o simulador que faz parte do ambiente EWS, utilizado na implementação.

A implementação do CCR e do SACF em ESTELLE forneceram, no entanto, uma especificação das implementações e dos relacionamentos entre estes elementos e a camada de Aplicação e Apresentação, segundo as idéias do capítulo 2, em uma Técnica de Descrição Formal. Procurou-se especificar o CCR e o SACF de maneira independente de características do ambiente de execução

(o SISDI-OSI). Análises sobre a implementação semi-automática de protocolos utilizando compiladores para ESTELLE aparecem em [Boc87] e [Vuo88].

O Ambiente EWS

O ambiente EWS (*ESTELLE Workstation*) [EWS89] é um conjunto de ferramentas para o processamento de especificações em ESTELLE, tanto para a geração de implementações (códigos para a execução de uma especificação), quanto para simulação. O EWS faz parte do *Projeto Esprit 1265*. As ferramentas deste ambiente são:

1. o *EWSEDIT*, que é um editor de texto orientado à sintaxe de ESTELLE, para a geração das especificações;

2. o *EWSTRANS*, que analisa a sintaxe e a semântica da especificação e gera um código intermediário, a ser processado por outras ferramentas;

3. o *EWSGEN*, que traduz a especificação em ESTELLE, a partir do código intermediário gerado por EWSTRANS, para um código em C que é executado de acordo com a semântica de ESTELLE. Nesta tese não faz parte a análise do código gerado. Algumas informações sobre como esta geração é feita podem ser obtidas em [EWS89] e uma descrição mais detalhada do código para um caso específico, em [Lin92];

4. um simulador, composto pelo *EWS Simulator Motor*, que é um núcleo de funções que executam a simulação, e pela *ESTELLE Simulator Interface*, que é uma interface gráfica para o usuário interagir com o núcleo e controlar a simulação; e

5. o *ESKIMO*, que é um núcleo de rotinas que são acopladas ao código em C gerado por EWSGEN, para que o código total para execução fique completo. Estas rotinas fornecem os aspectos dinâmicos de uma especificação ESTELLE.

Para as implementações desta tese só foram usados o EWSTRANS, o EWSGEN e o ESKIMO. O EWSTRANS, além da geração do código intermediário, gera uma listagem com informações sobre a compilação e uma listagem de referências cruzadas. Na primeira destas listagens aparecem os erros e avisos (*warnings*) que porventura tenham sido detectados na análise da especificação. A segunda listagem fornece informações sobre a especificação, úteis para documentação. São informações como as linhas em que cada objeto foi citado, uma descrição da hierarquia dos módulos, além de algumas estatísticas, como o número total de linhas, de canais usados, de pontos de interação, de transições, dentre outras.

O EWSTRANS segue a definição de ESTELLE com o *status* de *draft proposal* de 1987, que é a versão comentada em [Linn87]. Algumas restrições foram colocadas, em função de limitações de implementação, como número máximo para elementos nos conjuntos de estados, ou o fato de só se poder definir rotinas *PRIMITIVE* no nível mais alto da hierarquia de módulos. Não implementa prioridades. Como extensão da linguagem foram incluídos *comentários qualificados* (*qualifying comments*), para a possibilidade do uso de diretivas.

O ESKIMO fornece rotinas para a implementação de apenas um sistema ESTELLE, que corresponde a apenas uma instância de um módulo com classe *SYSTEM*. Há, no entanto, facilidades para que se possa definir mais de um destes sistemas, gerando-se mais de um processo a nível do sistema operacional.

5.3 A Implementação do CCR

A implementação do protocolo CCR foi feita como se os seus usuários e o SACF também fossem implementados em ESTELLE e fizessem todos parte de uma mesma especificação, comunicando-se diretamente através dos canais desta linguagem. Pretendia-se, com isto, obter, na especificação em ESTELLE, a independência em relação ao ambiente de execução.

Como o SISDI-OSI não funcionará desta maneira, ou seja, os outros módulos são implementados em outros processos (mesmo se forem implementados em ESTELLE) e a comunicação se fará através de filas de mensagens associadas a estes processos e de uma área de memória compartilhada, o esquema de implementação do CCR é o que aparece na figura 5.3.

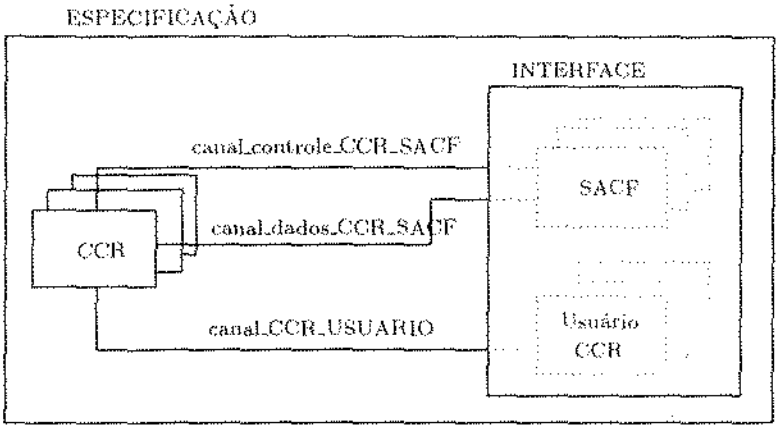

Figura 5.3: Estrutura de implementação do CCR em ESTELLE

Nesta figura, há três módulos ESTELLE: o módulo *CCR*, o módulo *INTERFACE* e o módulo de *ESPECIFICAÇÃO* (*Specification*), estando este último presente em todas as especificações em ESTELLE. O módulo *CCR* implementa a máquina de estados deste protocolo, para todas as quatro funções de seus usuários, ou seja, superior e subordinado de um ramo durante a transmissão normal de dados da ação atômica e superior e subordinado durante a fase de recuperação. No documento [ISO9805], que especifica o protocolo do CCR, cada uma destas funções é descrita por uma tabela de estado. Na implementação, no entanto, estas máquinas foram unidas em uma só.

Haverá uma instância do módulo *CCR* para cada associação que fizer uso deste protocolo. Esta instância será criada no momento de criação do SAO pelo SACF, quando as informações iniciais sobre o SAO são enviadas pelo SACF ao CCR, e termina quando o SAO for destruído.

O módulo *INTERFACE* será usado para compatibilizar a implementação do CCR com as demais do SISDI-OSI, que podem ser feitas em outras linguagens. Este módulo converte as mensagens recebidas na fila externa do processo em mensagens ESTELLE para serem enviadas nos canais usados na implementação (estes canais aparecem na figura 5.3) e, de modo inverso, converte as mensagens recebidas dos canais para os formatos das mensagens do SISDI-OSI, para serem enviadas para os outros processos. Estas conversões são necessárias porque a comunicação através dos canais de ESTELLE se dá de modo análogo a chamadas de procedimentos de linguagens como PASCAL, com os parâmetros da interação passados como se fossem argumentos de chamada de subrotinas.

A comunicação através das filas implica em manipular áreas da memória compartilhada, tratando os dados segundo os formatos estabelecidos pelo compilador ASN.1 (ver capítulo 3).

Observe que os canais de comunicação do CCR, que, de acordo com a figura 5.1, deveriam ser para se comunicar com módulos que representassem seu usuário e o SACF (componentes com que interage), são, no entanto, usados na implementação para se comunicar com o módulo *INTERFACE*. Este módulo simula a existência das várias instâncias destes componentes (que aparecem tracejadas na figura 5.3) e armazena, para cada instância do módulo *CCR*, os identificadores da fila do processo usuário e do SACF usado. Desta maneira a especificação do protocolo CCR pôde ser feita como se seus pontos de interação fossem realmente conectados a pontos de interação dos módulos com que interage, como se eles existissem. Isto resultou em uma especificação mais independente do ambiente específico em que foi implementada (SISDI-OSI).

O módulo *INTERFACE* controla ainda a criação das instâncias do módulo *CCR*, juntamente com o módulo *ESPECIFICAÇÃO*, e terá apenas uma invocação, que fará as conversões para todas as instâncias do módulo *CCR*. Esta invocação é criada assim que o processo CCR é posto para executar, ou seja, na fase de "inicialização" do módulo *ESPECIFICAÇÃO*, e dura enquanto o processo CCR ficar ativo.

O módulo *ESPECIFICAÇÃO*, pai dos outros módulos, é o responsável por criar e apagar instâncias do módulo *CCR*, além da criação da única instância do módulo *INTERFACE*, e conectar seus pontos de interação. Em ESTELLE somente uma instância de módulo pai pode criar e eliminar instâncias de módulos filhos.

5.3.1 A Implementação do Módulo *CCR*

Nesta seção será discutida a especificação em ESTELLE que gera o código para o protocolo CCR no SISDI-OSI.

Cabeçalho do Módulo *CCR*

O cabeçalho do módulo *CCR* aparece na figura 5.4. Neste cabeçalho estão:

1. um ponto de interação para troca de primitivas de serviços com o usuário (*pa_USUARIO*);
2. um ponto de interação para troca de primitivas de controle com o SACF (*pa_SACF_controle*);
3. um ponto de interação para enviar para e receber do SACF APDUs do CCR (*pa_SACF_dados*); e
4. uma variável exportada chamada *ext_terminou*, do tipo *BOOLEAN*. Esta variável é usada para se controlar o término de instâncias de módulos. O seu uso será comentado na seção 5.3.2.

Os módulos foram especificados como sendo da classe PROCESS. A atribuição de classe a módulos folhas da árvore de hierarquia de módulos não altera em nada o funcionamento de um sistema. Estes atributos influenciam a escolha de transições das instâncias de módulos filhos a serem executadas, e módulos folhas não possuem módulos filhos. A atribuição da classe só tem que estar de acordo com a classe de seu módulo pai. O módulo *ESPECIFICAÇÃO*, pai do módulo *CCR*, foi declarado como SYSTEMPROCESS.

```
MODULE Modulo_CCR PROCESS;
  IP
     pa_USUARIO        : canal_CCR_usuario (CCR) COMMON QUEUE;
     pa_SACF_controle  : canal_controle_CCR_SACF (CCR) COMMON QUEUE;
     pa_SACF_dados     : canal_dados_CCR_SACF (CCR) COMMON QUEUE;

  EXPORT
     ext_terminou : BOOLEAN;
END;
```

Figura 5.4: Cabeçalho do módulo *CCR*

Canais

Os canais usados para a troca de mensagens com os componentes com os quais o CCR interage aparece na figura 5.3. Como comentado anteriormente, os módulos *SACF* e *USUÁRIO DO CCR* desta figura são representados na implementação pelo módulo *INTERFACE*, por isto estão representados em linhas tracejadas. Eles não estão realmente implementados. Os canais são definidos para pontos de interação entre o módulo *CCR* e o módulo *INTERFACE*. As conversões feitas por este último módulo, comentadas anteriormente, são das mensagens recebidas e a serem enviadas para estes canais.

O canal *canal_CCR_usuario* é usado para a troca de primitivas de serviço entre o módulo *CCR* e seu usuário. A definição deste canal aparece na figura 5.5 (para algumas interações foram usadas reticências na parte de parâmetros, por economia de espaço).

Na definição deste canal estão todas as primitivas (interações, segundo a nomenclatura de ESTELLE) que o módulo *CCR* pode receber de e enviar a seu usuário. As primitivas do CCR e seus parâmetros foram apresentadas na seção 4.2. Observe que quando um parâmetro é opcional há um campo do tipo *Presente_ou_Ausente* antes dele na definição da primitiva. O nome deste parâmetro é formado acrescentando-se *op*, de *opcional*, no meio ou no final do nome do parâmetro a que está relacionado. O tipo *Presente_ou_Ausente* é um tipo inteiro. Se um parâmetro deste tipo possuir o valor da constante *PRESENTE*, significa que o parâmetro seguinte está sendo usado; se tiver o valor da constante *AUSENTE*, significa o contrário, e o valor do campo seguinte não é para ser considerado. Valores para estas constantes são definidos globalmente para o sistema. Estes parâmetros não foram definidos como do tipo *BOOLEAN* para se garantir a compatibilidade entre os códigos gerados para este protocolo e os de outros componentes do SISDI-OSI que usarão os valores deste campo, pelo fato de eles poderem ser implementados usando-se outras linguagens.

Foram definidas interações para a emissão conjunta de primitivas, como mencionado na seção 4.2. Estas interações são as C_COMMIT_req_C_BEGIN_req e C_ROLLBACK_req_C_BEGIN_req, para a emissão conjunta, respectivamente, das primitivas C_COMMIT.req e C_BEGIN.req e das C_ROLLBACK.req e C_BEGIN.req pelo usuário, e as interações C_COMMIT_ind_C_BEGIN_ind e C_ROLLBACK_ind_C_BEGIN_ind, para, respectivamente, a emissão das primitivas C_COMMIT.ind e C_BEGIN.ind e das C_ROLLBACK.ind e C_BEGIN.ind pelo CCR. Estas primitivas contêm a união dos parâmetros das primitivas isoladas. Os parâmetros de dados de usuário possuem um sufixo para indicar a qual das primitivas emitidas juntas eles pertencem.

```
CHANNEL canal_CCR_usuario (usuario, CCR);
 BY usuario :
  C_BEGIN_req(id_acao_atomica     :Tipo_Id_Acao_Atomica;
              id_ramo_acao_atomica:Tipo_Id_Ramo_Acao_Atomica;
              dados_usuario_op    :Presente_ou_Ausente;
              dados_usuario       :Tipo_Dados_Usuario );
  C_BEGIN_rsp(dados_usuario_op    :Presente_ou_Ausente;
              dados_usuario       :Tipo_Dados_Usuario );
  C_PREPARE_req( ... );
  C_READY_req( ... );
  C_COMMIT_req( ... );
  C_COMMIT_rsp( ... );
  C_ROLLBACK_req( ... );
  C_ROLLBACK_rsp( ... );
  C_RECOVER_req(id_acao_atomica     :Tipo_Id_Acao_Atomica;
                id_ramo_acao_atomica:Tipo_Id_Ramo_Acao_Atomica;
                estado_recuperacao  :Tipo_Estado_Recup_req_ind;
                dados_usuario_op    :Presente_ou_Ausente;
                dados_usuario       :Tipo_Dados_Usuario );
  C_RECOVER_rsp( ... );
  C_COMMIT_req_C_BEGIN_req(dados_usuario_op_c_commit:Presente_ou_Ausente;
                           dados_usuario_c_commit   :Tipo_Dados_Usuario;
                           id_acao_atomica          :Tipo_Id_Acao_Atomica;
                           id_ramo_acao_atomica     :Tipo_Id_Ramo_Acao_Atomica;
                           dados_usuario_op_c_begin :Presente_ou_Ausente;
                           dados_usuario_c_begin    :Tipo_Dados_Usuario );
   C_ROLLBACK_req_C_BEGIN_req( ... );
 BY CCR :
   C_BEGIN_ind( ... );
   C_BEGIN_cnf( ... );
   C_PREPARE_ind( ... );
   C_READY_ind( ... );
   C_COMMIT_ind( ... );
   C_COMMIT_cnf( ... );
   C_ROLLBACK_ind( ... );
   C_ROLLBACK_cnf( ... );
   C_RECOVER_ind( ... );
   C_RECOVER_cnf( ... );
   C_COMMIT_ind_C_BEGIN_ind( ... );
   C_ROLLBACK_ind_C_BEGIN_ind( ... );
```

Figura 5.5: Canal para troca de primitivas entre o CCR e seu usuário

As definições dos dois outros canais, usados para as interações com o SACF, aparecem na figura 5.6.

```
CHANNEL canal_dados_CCR_SACF (CCR,SACF);
 BY CCR, SACF :
   C_BEGIN_RI ( apontr_info : Tipo_Descritor );
   C_BEGIN_RC ( apontr_info  :  Tipo_Descritor );
   C_PREPARE_RI ( apontr_info  :  Tipo_Descritor );
   C_READY_RI ( apontr_info  :  Tipo_Descritor );
   C_COMMIT_RI ( apontr_info  :  Tipo_Descritor );
   C_COMMIT_RC ( apontr_info  :  Tipo_Descritor );
   C_ROLLBACK_RI ( apontr_info  :  Tipo_Descritor );
   C_ROLLBACK_RC ( apontr_info  :  Tipo_Descritor );
   C_RECOVER_RI ( apontr_info  :  Tipo_Descritor );
   C_RECOVER_RC ( apontr_info  :  Tipo_Descritor );
   C_COMMIT_RI_C_BEGIN_RI ( apontr_c_commit_ri : Tipo_Descritor;
                            apontr_c_begin_ri : Tipo_Descritor );
   C_ROLLBACK_RI_C_BEGIN_RI ( apontr_c_rollback_ri : Tipo_Descritor;
                              apontr_c_begin_ri : Tipo_Descritor );
   C_ROLLBACK_RC_C_BEGIN_RI ( apontr_c_rollback_rc : Tipo_Descritor;
                              apontr_c_begin_ri : Tipo_Descritor );

CHANNEL canal_controle_CCR_SACF (CCR,SACF);
 BY SACF :
   C_TERMINA_INVOCACAO;
   C_TITULO_AE_REMOTA ( titulo_AE : Tipo_Titulo_AE );
 BY CCR :
   C_ERRO_DE_PROTOCOLO_REMOTO ( apdu : Tipo_APDUs_CCR );
   C_ERRO_DE_PROTOCOLO_LOCAL ( prim : Tipo_Primitivas_CCR );
```

Figura 5.6: Canais para comunicação entre o CCR e o SACF

O canal *canal_dados_CCR_SACF* é usado para a troca de APDUs entre o CCR e o SACF. Observe que há interações especiais para a indicação de APDUs concatenadas (por exemplo, *C_COMMIT_RI_C_BEGIN_RI*) e que as APDUs podem ser enviadas nos dois sentidos (estão associadas aos dois tipos de *funções* definidas no cabeçalho do canal).

O único parâmetro das interações usadas para transmissão de APDUs entre o CCR e o SACF (*apontr_info*) funciona como um apontador. Este apontador indica onde, no *buffer*, os campos da APDU estão armazenados. Os tipos apontadores tiveram que ser tratados de modo especial, pois ESTELLE não permite que parâmetros de interações sejam apontadores. O que se fez foi considerar cada apontador como um tipo inteiro (*Tipo_Descritor*) e rotinas que implementam o *buffer* na área compartilhada convertem este inteiro para um apontador e vice-versa.

A implementação do *buffer* foi feita de modo que estas conversões são triviais. Como ele foi implementado como um vetor de *bytes*, o inteiro correspondente a um apontador é o índice deste

vetor que está no endereço armazenado no apontador e vice-versa. Esta violação às regras normais da linguagem foi feita porque se deseja que o código em ESTELLE especifique realmente como as APDUs são tratadas, ou seja, armazenadas em um *buffer* e acessadas através de apontadores. A especificação feita em ESTELLE do protocolo CCR é destinada a implementação, e procurou-se especificar um modo bastante comum de tratamento das APDUs em implementações [Svo89].

Estas rotinas de conversão são exemplos de rotinas declaradas como PRIMITIVE, o que quer dizer que os seus códigos não se encontram na especificação ESTELLE, mas sim em outros arquivos, implementadas em outra linguagem (C, no caso) e *link-editadas* com o código em C gerado a partir da especificação ESTELLE.

O canal *canal_controle_CCR_SACF* é usado para a troca de interações de controle entre o CCR e o SACF. No sentido do CCR para o SACF, estas interações são usadas para indicar condições de erro de protocolo. A interação *C_ERRO_DE_PROTOCOLO_REMOTO* indica a recepção de uma APDU emitida pelo sistema remoto fora da seqüência normal de eventos permitida pelo protocolo. A interação *C_ERRO_DE_PROTOCOLO_LOCAL* indica a emissão pelo usuário local do CCR (no mesmo sistema) de uma primitiva fora da seqüência normal de eventos do protocolo. Esta interação pode ser usada na fase de testes dos protocolos usuários do CCR e pode ser eliminada posteriormente, quando estiverem funcionando corretamente. Estas interações possuem, cada uma, um parâmetro que indica o evento (APDU ou primitiva) que causou o erro.

No sentido do SACF para o CCR há uma interação para solicitar o término da instância do módulo *CCR* (*C_TERMINA_INVOCACAO*) e outra para informar o título da AE remota (*C_TITULO_AE_REMOTA*). Este título é necessário ao CCR para preencher o parâmetro *identificador do ramo da ação atômica*, da primitiva C-BEGIN.ind, e é transmitido ao CCR assim que sua instância para uso no SAO for criada. O término de uma instância do módulo *CCR* é solicitado quando o SAO ao qual a instância pertence for ser destruído. Isto pode acontecer, por exemplo, no término normal de uso da associação ou quando a associação se rompe, devido a uma falha de comunicação, como uma detecção de erro de protocolo em uma camada inferior.

Observe que a mensagem enviada pelo SACF para a criação do SAO, de acordo com a figura 5.1, informa, além do título da AE remota, dois identificadores de filas. O envio da interação *C_TITULO_AE_REMOTA* que aparece na figura 5.6, é feito logo após o recebimento desta mensagem do SACF. Os identificadores não são necessários ao módulo *CCR* porque a especificação feita, como comentado anteriormente, assume que os pontos de interação seriam diretamente conectados a pontos de interação dos módulos com os quais interage, como se fossem especificados em ESTELLE. Os identificadores de fila, portanto, são mantidos somente pelo módulo *INTERFACE*, para que possa saber para qual processo as mensagens devem ser enviadas, não sendo, com isto, passados em interações.

Tipos e Funções Externas

Os tipos definidos para a implementação do módulo *CCR* são traduções para PASCAL das estruturas de dados em C geradas pelo compilador ASN.1 e de alguns tipos usados no sistema como um todo, como identificadores de filas, além de alguns tipos auxiliares, que apenas aumentam a clareza da especificação. As conversões para PASCAL dos tipos usados para a especificação do CCR foram simples de ser realizadas. Como comentado no capítulo 3, os tipos dos parâmetros de primitivas de serviços são derivados dos tipos dos campos correspondentes nas APDUs.

As funções de interação com o ambiente de execução são declaradas como PRIMITIVE. Para a

implementação do CCR foram declaradas como PRIMITIVE funções para realizar operações sobre filas (ler, escrever, etc.) e para realizar operações sobre a área compartilhada (montar primitivas no *buffer*, ler parâmetros de primitivas e campos de APDUs, alocar e desalocar áreas do *buffer*, etc.).

O Corpo do Módulo

O corpo do módulo *CCR* implementa a união das quatro máquinas de estados que aparecem em [ISO9805], uma para cada tipo de função do usuário do CCR : superior e subordinado durante uma execução normal da ação atômica e superior e subordinado para recuperação. A figura 5.7 esquematiza esta união. A união das máquinas resultou em uma máquina de estados genérica, que contém todos os estados de cada uma das máquinas, mas mantendo apenas um estado inicial, que corresponde à união dos estados iniciais de cada máquina unida. Quando a ação de cada máquina terminar, a máquina global retorna a este estado (representado, na figura, pelas linhas com setas que saem dos retângulos e vão ao círculo que representa o estado inicial). O término da instância está associado ao fim da associação, não ao fim da ação da máquina de estado para uma das funções.

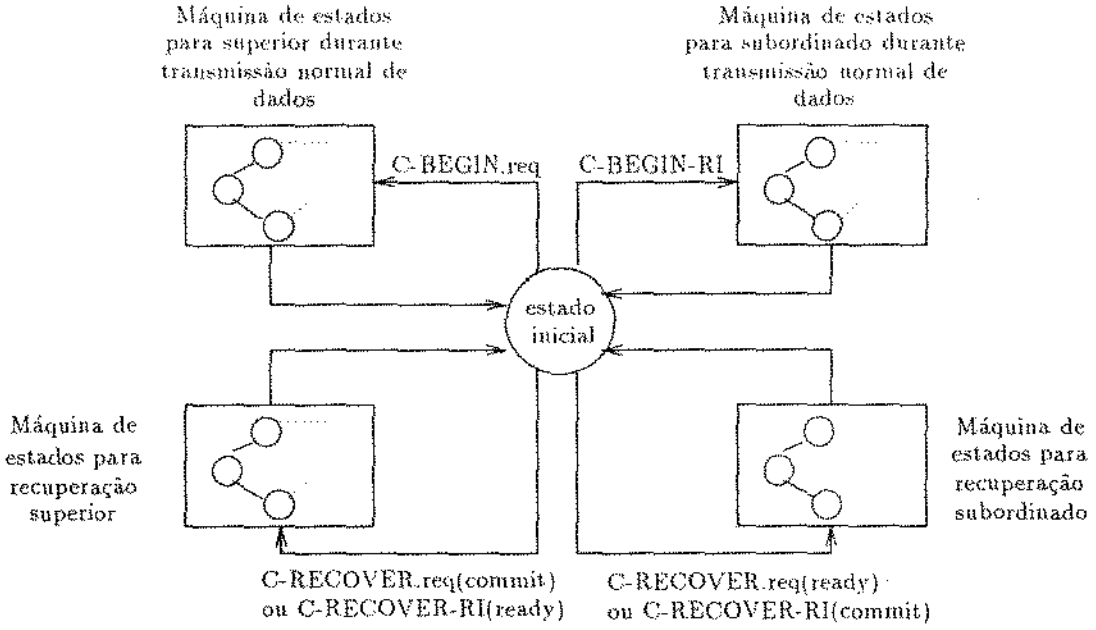

Figura 5.7: Máquina de estados implementada

O estado inicial serve para identificar qual é o tipo de função a ser desempenhado e isto é feito de acordo com o evento que chegar. Se o primeiro evento for uma primitiva C-BEGIN.req, a máquina de estados sofrerá transições entre os estados para a função de superior durante a transmissão normal de dados; se for uma APDU C-BEGIN-RI, as transições serão entre os estados para a função de subordinado durante a transmissão normal de dados; se for uma primitiva C-RECOVER.req, com parâmetro *estado_recuperacao* (figura 5.5) com valor *COMMIT*, ou uma APDU C-RECOVER-RI, com o campo de estado de recuperacao (obtido através do campo apontador desta APDU, na

figura 5.6) com valor *READY*, os estados serão os de recuperação superior; se for uma primitiva C-RECOVER.req, com o parâmetro *estado_recuperacao* com valor *READY*, ou uma APDU C-RECOVER-RI, com o campo de estado de recuperacao com valor *COMMIT*, os estados serão os de recuperação subordinado. Qualquer outra primitiva ou APDU que chegar neste estado causará um erro de protocolo.

As partes da máquina de estado implementada que correspondem a cada uma das funções foram derivadas das máquinas para estas funções especificadas no documento do protocolo CCR [ISO9805]. As partes relativas a recuperação foram derivadas diretamente. As correspondentes às funções de superior e subordinado para a transmissão normal de dados, no entanto, sofreram algumas alterações.

As alterações feitas são devidas ao fato de a comunicação entre o processo CCR e seus usuários no SISDI-OSI não ser imediata. A especificação do protocolo no documento [ISO9805], no entanto, implicitamente assume que seja. Pelo fato de os componentes do SISDI-OSI serem implementados cada um como um processo UNIX separado e haver filas para a comunicação entre eles, pode acontecer de uma mensagem emitida por um componente somente ser processada pelo componente receptor algum tempo depois, sendo que, neste interim, os processos emissor e receptor podem tratar outras mensagens, que podem alterar o estado da associação.

Com isto, algumas situações podem ocorrer que aparentam um erro de protocolo. Este seria o caso, por exemplo, se chegasse uma APDU C-ROLLBACK-RI, vinda do sistema remoto, enquanto o usuário do CCR emitisse a primitiva C-PREPARE.req. Na especificação do protocolo no documento [ISO9805], se a APDU for tratada primeiro, o ASE CCR poderia assumir um erro de protocolo ao receber a primitiva, já que tudo se passa como se o usuário teria recebido a primitiva C-ROLLBACK.ind e então teria enviado a primitiva C-PREPARE.req (suposição de comunicação imediata). Pelo modo como os componentes foram implementados, no entanto, o usuário teria emitido a primitiva C-PREPARE.req antes de receber a C-ROLLBACK.ind, mas o CCR só a teria tratado depois da APDU C-ROLLBACK-RI.

Estes casos foram detectados nas tabelas de estados de superior e subordinado durante transmissão normal de dados e ocorrem para as APDUs C-READY-RI e C-ROLLBACK-RI. Nas tabelas referentes a recuperação estes casos não ocorrem. A tabela 5.1 mostra a tabela de estados para a função de superior durante transmissão normal de dados, após as medidas tomadas para o tratamento destes casos. Nesta tabela o estado que aparece em cada célula indica o próximo estado que a máquina ficará, uma vez tendo recebido o evento associado à linha da célula, estando a máquina no estado associado à coluna da célula. As medidas foram :

1. descarte da primitiva emitida pelo usuário. Nestas situações, para o caso do CCR, a APDU emitida pelo usuário remoto faz com que a primitiva do usuário não precise mais ser tratada. Por exemplo, no caso acima, já que o usuário remoto enviou uma APDU C-ROLLBACK-RI, não há mais necessidade do sistema local requisitar o voto do sistema remoto (emitindo C-PREPARE.req);

2. novos estados foram criados para controlar o seqüenciamento das primitivas emitidas pelo usuário nestas situações. Isto foi feito, por exemplo, para o estado $A5$ da tabela de estados. Este estado corresponde ao recebimento da APDU C-READY-RI. Pelo fato de o CCR poder receber a primitiva C-PREPARE.req mesmo após já ter tratado esta APDU, um novo estado $A5'$ foi criado. Este estado possui as mesmas transições que o estado $A5$, com exceção de que, neste último estado, a chegada da primitiva C-PREPARE.req causa uma transição para

o novo estado, enquanto que no estado *A5'* causa um erro (a primitiva C-PREPARE.req já foi emitida). Os estados criados possuem apóstrofes em seus nomes;

3. como novos estados foram criados, algumas transições para o estado original (por exemplo, *A5*) precisariam ser atualizadas para o novo estado criado. Por exemplo, as transições dos estados *A3* e *A4* causadas pela recepção da APDU C-READY-RI, originalmente estão especificadas para o estado *A5*. Devem, no entanto, passar a indicar uma transição para o novo estado *A5'*, pois, para a máquina de estados estar em um dos estados *A3* e *A4*, significa que o usuário já emitiu a primitiva C-PREPARE.req.

Estas modificações, no entanto, passaram a permitir certas seqüências de eventos que são erros de seqüenciamento de acordo com o protocolo, mas que passam a não ser mais detectadas. Isto acontece, por exemplo, no caso comentado anteriormente, da APDU C-ROLLBACK-RI e da primitiva C-PREPARE.req, se o usuário realmente tivesse emitido a primitiva C-PREPARE.req depois de receber a primitiva C-ROLLBACK.ind. Estes casos, no entanto, não causam problema, pois, segundo o item 1 acima, a primitiva do usuário é descartada, não enviando, portanto, uma APDU fora de seqüência para o sistema remoto.

Nas células não vazias onde o estado aparece entre parênteses, a primitiva associada à linha da célula é descartada. Nas células não vazias em que isto não acontece, a APDU ou primitiva é gerada de acordo com a tabela 4.1.

As células vazias que restaram na tabela correspondem a erros de protocolo. Algumas destas células correspondem, no entanto, a uma combinação de evento e estado que nunca ocorreria, caso o sistema local fosse corretamente implementado. Este seria o caso, por exemplo, da interseção entre o evento C-BEGIN-RC e o estado *A9*. Este estado representa que foi recebida uma APDU C-ROLLBACK-RI, que gera a primitiva C-ROLLBACK.ind, e que se está esperando a confirmação (C-ROLLBACK.rsp). O fato de esta APDU ser transmitida em uma primitiva do serviço P-RESYNCHRONIZE impede que o usuário remoto envie a APDU C-BEGIN-RC, que é transmitida em primitivas do serviço P-TYPED-DATA. Não precisaria, portanto, haver tratamento para tais células.

Na implementação atual, no entanto, deixou-se que se detectassem erros de protocolo mesmo nestes casos, pois pode ser útil no teste de outras implementações do sistema. As alterações na especificação para se tratar ou não estas células é simples de se fazer.

As alterações necessárias na máquina de estado para a função de subordinado durante transmissão normal de dados são obtidas seguindo os mesmos passos das alterações comentadas nesta seção para a máquina de estados para a função de superior durante transmissão normal de dados.

Estados

Como a máquina de estados implementada no módulo *CCR* é a união das máquinas de estado para cada uma das funções, os seus estados também serão a união dos estados de cada uma destas máquinas, com exceção de que só haverá um estado inicial. Este estado inicial corresponde a uma combinação dos estados iniciais de cada máquina, como comentado anteriormente. A cláusula *STATE* do módulo *CCR*, portanto, contém simplesmente a declaração de todos estes estados.

Além dos estados simples, conjuntos de estados também foram definidos, para se fazer controle de erros. Para cada evento que deve ser tratado pelo módulo (primitiva ou APDU) e para cada máquina de estados para uma função do usuário do CCR em que este evento puder ocorrer

	I	*A1*	*A2*	*A3*	*A4*	*A5*	*A5'*	*A6*
C-BEGIN.req	A1							
C-BEGIN.RC		A2		A4				
C-PREPARE.req		A3	A4			(A5')		
C-READY-RI		A5	A5	A5'	A5'			
C-COMMIT.req						A6	A6	
C-COMMIT-RC								I
C-ROLLBACK.req		A7	A7	A7	A7	A8	A8	
C-ROLLBACK-RC								
C-ROLLBACK-RI		A9	A9	A9'	A9'			
C-ROLLBACK.rsp								
C-COMMIT.req + *C-BEGIN.req*						A10	A10	
C-ROLLBACK.req + *C-BEGIN.req*		A11	A11	A11	A11	A13	A13	

	A7	*A8*	*A9*	*A9'*	*A9"*	*A10*	*A11*	*A12*	*A13*
C-BEGIN.req									
C-BEGIN.RC									
C-PREPARE.req			(A9')						
C-READY-RI									
C-COMMIT.req									
C-COMMIT-RC						A1			
C-ROLLBACK.req			(A9")	(A9")					
C-ROLLBACK-RC	I	I					A1		A1
C-ROLLBACK-RI	A9"						A12		
C-ROLLBACK.rsp			I	I	I			A1	
C-COMMIT.req + *C-BEGIN.req*									
C-ROLLBACK.req + *C-BEGIN.req*			(A9")	(A9")					

Tabela 5.1: Tabela de estados para a função de superior durante transmissão normal de dados

foi definido um conjunto indicando os estados nos quais este evento causa um erro de protocolo. Há também conjuntos de estados que contêm os estados definidos para cada tipo de função desempenhada pelo usuário do CCR. Com estes conjuntos de estados pode-se especificar todos os estados, na máquina global, em que cada evento causa um erro de protocolo. A figura 5.8 ilustra a definição de alguns estados, especificamente, o estado inicial (*estado_I*) e o conjunto de estados para a função de superior durante transmissão normal dos dados (*estado_A1*, *estado_A2*, etc.), e os conjuntos de estados definidos para tratar os erros associados ao evento C-ROLLBACK.req para a função de superior e subordinado durante a transmissão normal de dados (conjunto de estado *estados_erro_C_ROLLBACK_req_sup* e *estados_erro_C_ROLLBACK_req_sub*, respectivamente). Cada apóstrofe do nome do estado na tabela 5.1 se transforma na letra "l" no nome do estado na figura 5.8. Estes estados e conjuntos de estados, com exceção do conjunto *estados_erro_C_ROLLBACK_req_sub*, foram tirados da tabela 5.1. As reticências indicam partes em que outros estados e conjuntos de estados existem, mas não são mostrados.

```
STATE

  { Estado inicial }

    estado_I,

  { Estados para a funcao de superior durante transmissao normal de dados : }

    estado_A1, estado_A2, estado_A3, estado_A4,  estado_A5, estado_A5l,
    estado_A6,  estado_A7, estado_A8, estado_A9, estado_A9l, estado_A9ll,
    estado_A10, estado_A11, estado_A12, estado_A13,
       ...

STATESET
       ...

    estados_erro_C_ROLLBACK_req_sup = [estado_I,   estado_A6,  estado_A7,
                         estado_A8, estado_A9ll, estado_A10, estado_A11,
                         estado_A12, estado_A13];

    estados_erro_C_ROLLBACK_req_sub = [estado_I,  estado_B5,  estado_B6,
                         estado_B7, estado_B8ll, estado_B9, estado_B10,
                         estado_B8111, estado_B1111];
       ...
```

Figura 5.8: Estados e conjuntos de estados

Parte de Inicialização

A parte de inicialização do módulo *CCR* contém apenas a atribuição do valor *FALSE* à variável *ext_terminou* e a atribuição do valor *DESCRITOR_NULO* à variável *pt_c_begin_ri_armaz* (figura 5.9). O *DESCRITOR_NULO* é o valor inteiro que corresponde a um valor *NULL* para um apontador. A variável *ext_terminou* serve, como será comentado na seção 5.3.3, para uma instância solicitar seu término ao módulo *ESPECIFICAÇÃO*. A variável *pt_c_begin_ri_armaz* serve para armazenar uma cópia da APDU C-BEGIN-RI, que pode ter que ser retransmitida, se houver colisão, como comentado na seção 4.6.2, entre a concatenação das APDUs C-ROLLBACK-RI + C-BEGIN-RI e uma apdu C-ROLLBACK-RI. A cláusula *TO* é usada na parte de inicialização para indicar o estado inicial da máquina de estados, que no caso é *estado_I*.

```
INITIALIZE
 TO estado_I
  BEGIN
   ext_terminou := FALSE;
   pt_c_begin_ri_armaz := DESCRITOR_NULO;
  END;
```

Figura 5.9: Inicialização do módulo *CCR*

Parte de Transições

A grande maioria das transições do módulo *CCR* só utiliza as cláusulas WHEN e FROM, ou seja, tratam apenas de chegada de interações, de acordo com os estados em que estão. Apenas algumas poucas utilizam a cláusula PROVIDED, usada para especificar uma expressão condicional para que a transição seja habilitada.

Para o módulo *CCR* há transições para:

1. tratar cada evento (primitiva ou APDU) que o ASE pode receber em cada uma das funções de seu usuário. Haverá uma transição para cada célula não vazia da tabela de estados (algumas transições estão, no entanto, escritas de modo *aninhado*);
2. detectar erros de protocolos cometidos pelo usuário do CCR local ou pelo remoto;
3. tratar uma interação que fornece o título da AE remota;
4. tratar o pedido de término de instância do módulo *CCR*, solicitado pelo SACF.

As transições para os itens 3 e 4 acima aparecem na figura 5.10. A transição que trata da recepção do título da AE remota simplesmente armazena este título em uma variável local (*titulo_AE_remota*). A segunda transição, que trata o pedido de término de invocação, simplesmente atribui o valor *TRUE* à variável exportada *ext_terminou*. Observe que as interações vêm no ponto de interação com o SACF para a transmissão de primitivas de controle (*pa_SACF_controle*).

Para se gerarem as transições para as células não vazias das tabelas de estados do protocolo, que correspondem às transições referenciadas no item 1 acima, para cada evento, primitiva ou APDU,

```
TRANS

  { transicao para a recepcao das informacoes iniciais : }

   WHEN pa_SACF_controle.C_TITULO_AE_REMOTA
    FROM estado_I
     BEGIN
      titulo_AE_remota := titulo_AE;
     END;

  { transicao para o tratamento de pedido de termino de invocacao : }

   WHEN pa_SACF_controle.C_TERMINA_INVOCACAO
    BEGIN
     ext_terminou := TRUE;
    END;
```

Figura 5.10: Transições de controle do módulo *CCR*

foram tomadas as interseções válidas (não vazias) na linha do evento e foi escrita uma transição para cada conjunto de estados para os quais as ações a serem executadas e o estado resultante são os mesmos. Para ilustrar, tomou-se como exemplo o evento que corresponde à emissão da primitiva C-ROLLBACK.req pelo usuário do CCR, para a função de superior durante transmissão normal de dados. Para este evento e esta função, cuja linha da tabela de estados está mostrada na tabela 5.1 (sétima linha), as transições criadas são as que aparecem na figura 5.11.

Observe que a primitiva vem do usuário do CCR (ponto de interação *pa_USUARIO*, definido na figura 5.4) e que a cláusula WHEN é herdada pelas transições seguintes. Os parâmetros da interação C-ROLLBACK.req (*dados_usuario_op* e *dados_usuario*) são referenciados dentro da transição. Em uma transição com a cláusula WHEN os parâmetros da interação podem ser acessados pelos identificadores que têm na definição da interação. A função *monta_buffer_apdu_dados_usuar*, assim como outras, é declarada como PRIMITIVE. Esta função monta no *buffer* compartilhado pelos processos uma APDU que possua somente os campos *dados_usuario_op* e *dados_usuario*. Esta APDU é estruturada da maneira mostrada no capítulo 3.

Para este evento criou-se uma transição para os estados *estado_A1*, *estado_A2*, *estado_A3* e *estado_A4* e outra para os estados *estado_A5* e *estado_A5l*. Embora as ações tomadas nas duas transições sejam as mesmas, ou seja, montar a APDU C-ROLLBACK-RI no *buffer* e enviá-la ao SACF, não se pôde criar uma transição única para todos estes estados, pois os estados resultantes não são os mesmos. Na primeira transição a máquina de estado irá para o estado *estado_A7*, enquanto na segunda irá para o estado *estado_A8* (cláusula *TO*).

A última transição da figura foi criada para atender a células da tabela de estados do protocolo que correspondem a situações, como as comentadas anteriormente, causadas pelo fato de a interação entre o CCR e seu usuário não ser imediata. Nestas situações as primitivas são descartadas. O caso desta transição da figura 5.11 corresponde ao usuário emitir uma primitiva C-ROLLBACK.req após

```
{ transicoes para C_ROLLBACK_req : }

WHEN pa_USUARIO.C_ROLLBACK_req
 FROM estado_A1, estado_A2, estado_A3, estado_A4
  TO estado_A7
   BEGIN
    pt_area_buffer := monta_buffer_apdu_dados_usuar (C_ROLLBACK_RI,
                          dados_usuario_op,dados_usuario);
    OUTPUT pa_SACF_dados.C_ROLLBACK_RI (pt_area_buffer);
   END;

 FROM estado_A5, estado_A51
  TO estado_A8
   BEGIN
    pt_area_buffer := monta_buffer_apdu_dados_usuar (C_ROLLBACK_RI,
                          dados_usuario_op,dados_usuario);
    OUTPUT pa_SACF_dados.C_ROLLBACK_RI (pt_area_buffer);
   END;

 FROM estado_A9, estado_A91
  TO estado_A911
   BEGIN
    { Descarta por causa da falta de sincronismo com o usuario }
    IF (dados_usuario_op = PRESENTE)
     THEN desaloca_arvore_apdus (dados_usuario);
   END;
```

Figura 5.11: Transições para C-ROLLBACK.req

o usuário remoto já ter enviado uma APDU C-ROLLBACK-RI (estados *A9* e *A9l*). O tratamento da primitiva não precisa ser feito, pois o tratamento da APDU já causa o retorno da ação atômica ao estado inicial. A máquina de estados vai para o estado *A9*".

O descarte da primitiva requer, no entanto, que as APDUs que vierem como dado de usuário, caso haja alguma, também sejam descartadas. Observe que a transição verifica a existência ou não de APDUs consultando o valor do parâmetro *dados_usuario_op*. Como comentado no início desta seção, este parâmetro terá o valor representado pela constante *PRESENTE* se houver APDU no parâmetro de dados de usuário, e o valor da constante *AUSENTE*, em caso contrário.

O problema de descartar esta APDU é que ela pode também conter um campo para dados de usuário e neste campo haver outra APDU. O mesmo pode acontecer também com esta última APDU, gerando uma lista de APDUs, cada uma sendo o valor de um campo da seguinte na lista, como ilustra a figura 5.12. Pela definição dos campos de dados de usuários de protocolos da Aplicação, como comentado no capítulo 3, poderia haver concatenação de APDUs como valores de dados de usuário de outras APDUs, ao invés de apenas uma só APDU. Com isto esta lista se expandiria em uma árvore de APDUs. A desalocação da APDU na última transição da figura 5.11 deve causar a desalocação de toda a lista ou árvore de APDUs.

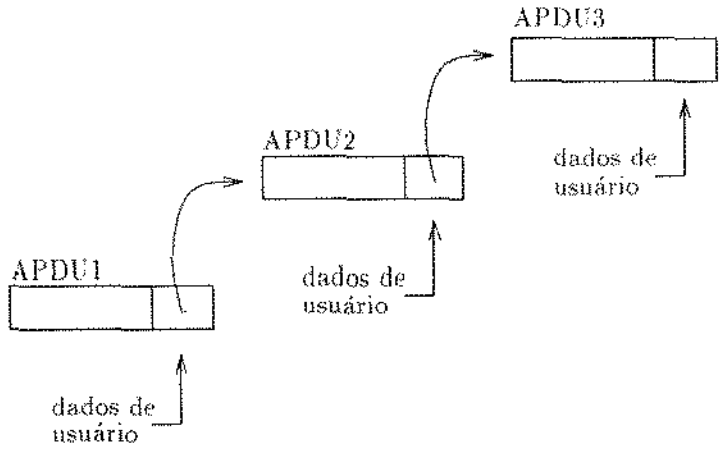

Figura 5.12: Lista de APDUs

Para se fazer esta desalocação na camada de Aplicação no SISDI-OSI adotou-se fazer uma rotina, acessada por todos os protocolos, que possui a função de tratar da desalocação desta árvore. Esta rotina, chamada *desaloca_arvore_apdus* recebe como parâmetro a raiz da árvore de APDUs que se quer desalocar e chama rotinas específicas de desalocação de cada tipo de APDU. Estas rotinas específicas são fornecidas pelo implementador de cada protocolo da Aplicação.

Para gerar as transições do item 2 acima, de detecção de erros de protocolo, foi feita uma transição para cada evento, que fica habilitada quando o evento é recebido em um estado inválido. Para exemplificar, a transição de detecção de erro para o mesmo evento, C-ROLLBACK.req, é a que aparece na figura 5.13.

Os estados nos quais a recepção da primitiva C-ROLLBACK.req é um erro são especificados na cláusula FROM. Estes estados são representados pelos conjuntos de estados:

```
{ transicao de deteccao de erro para C_ROLLBACK_req : }

FROM estados_erro_C_ROLLBACK_req_sup, estados_erro_C_ROLLBACK_req_sub,
     estados_recup_superior, estados_recup_subordinado
 WHEN pa_USUARIO.C_ROLLBACK_req
  BEGIN
   OUTPUT pa_SACF_controle.C_ERRO_DE_PROTOCOLO_LOCAL(C_ROLLBACK_req);
   IF (dados_usuario_op = PRESENTE)
    THEN desaloca_arvore_apdus (dados_usuario);
   ext_terminou := TRUE;
  END;
```

Figura 5.13: Transição para detecção de erro

1. *estados_erro_C_ROLLBACK_req_sup*, que contém todos os estados nos quais a interseção com a linha correspondente ao evento C-ROLLBACK.req é vazia na tabela de estados para a função de superior durante transmissão normal de dados (tabela 5.1). Este conjunto de estados foi mostrado na figura 5.8;

2. *estados_erro_C_ROLLBACK_req_sub*, que contém todos os estados nos quais a interseção com a linha correspondente ao evento C-ROLLBACK.req é vazia na tabela de estados para a função de subordinado durante transmissão normal de dados. Este conjunto de estados também foi mostrado na figura 5.8;

3. *estados_recup_superior*, que contém os estados da máquina de estados para a função de superior durante recuperação. Durante a recuperação a primitiva C-ROLLBACK.req não pode ser emitida; e

4. *estados_recup_subordinado*, que contém os estados da máquina de estados para a função de subordinado durante recuperação.

A ação especificada na transição é enviar a indicação de erro de protocolo para o SACF, que informará ao *USUÁRIO DO SAO*: verificar se há alguma APDU no parâmetro de dados de usuário da primitiva e, se houver, desalocá-la (e a possível lista ou árvore de APDUs); e atribuir o valor *TRUE* à variável *ext_terminou*, para que o módulo *ESPECIFICAÇÃO* elimine a instância do módulo *CCR*. Na indicação de erro de protocolo está informada qual primitiva causou o erro.

Um último comentário a fazer sobre as transições deste módulo é que, pelo fato de a APDU C-BEGIN-RI poder ter que ser retransmitida, como comentado anteriormente, quando se falou sobre a parte de inicialização deste módulo, a máquina de estados do CCR deve copiar a APDU C-BEGIN-RI enviada. Como pode haver, de modo análogo ao caso comentado acima para a desalocação de APDUs, uma árvore de APDUs com raiz no campo de dados de usuário desta APDU, houve a necessidade de se fazer uma rotina para a cópia destas árvores. Esta rotina é análoga à rotina *desaloca_arvore_apdus*. É acessada por todos os protocolos e, a partir da raiz da árvore, passada como parâmetro, invoca rotinas específicas de cópias de APDUs fornecidas pelos implementadores de cada protocolo da Aplicação.

5.3.2 O Módulo *INTERFACE*

Nesta seção será comentada genericamente a especificação em ESTELLE para o módulo *INTERFACE*.

O módulo *INTERFACE* possui na definição de seu cabeçalho, mostrada na figura 5.14, pontos de interação para trocar interações com as instâncias do módulo *CCR*, simulando os relacionamentos mostrados na figura 5.1. Este módulo, como mencionado anteriormente, simula a existência dos módulos *USUÁRIO DO CCR* e *SACF*, como se estes módulos e o módulo *CCR* tivessem todos sido implementados em ESTELLE, formando uma única especificação. Em seu cabeçalho há três vetores de pontos de interação, cada um para tratar um dos tipos de canais definidos para o módulo *CCR*, ou seja, para troca de primitivas de serviço com seu usuário e de primitivas de controle e APDUs com o SACF (observe que as funções declaradas para os pontos de interação são *usuario* e *SACF*).

```
MODULE Modulo_Interface PROCESS;
  IP
   pa_USUARIO_CCR: ARRAY [Tipo_Faixa_Associacoes] OF
                   canal_CCR_usuario (usuario) COMMON QUEUE;
   pa_SACF_CCR_controle: ARRAY [Tipo_Faixa_Associacoes] OF
                   canal_controle_CCR_SACF (SACF) COMMON QUEUE;
   pa_SACF_CCR_dados: ARRAY [Tipo_Faixa_Associacoes] OF
                   canal_dados_CCR_SACF (SACF) COMMON QUEUE;
  EXPORT
     ext_conectar     : Tipo_Fases_Conexao;
     ext_id_invoc     : INTEGER;
END;  { Modulo_Interface }
```

Figura 5.14: Cabeçalho do módulo *INTERFACE*

O tamanho de cada vetor é igual ao número máximo de associações que o SISDI-OSI permitir, pois pode ser que o CCR seja usado em todas as associações, necessitando, por isso, uma instância do módulo *CCR* para cada associação. O tipo *Tipo_Faixa_Associacoes* é definido da seguinte maneira (*NUMMAX_ASSOCIACOES* é uma constante que define o número máximo de associações que o SISDI-OSI suporta) :

Tipo_Faixa_Associacoes = 1..NUMMAX_ASSOCIACOES;

As variáveis exportadas, *ext_conectar* e *ext_id_invoc*, são usadas para a comunicação entre o módulo *INTERFACE* e o módulo *ESPECIFICAÇÃO*, para que este possa criar as instâncias do módulo *CCR*, como será comentado adiante.

Foi atribuída a classe *PROCESS* para o módulo *INTERFACE* para se compatibilizar com o atributo para o módulo *ESPECIFICAÇÃO*, de modo análogo ao caso do CCR.

O funcionamento dos protocolos do SISDI-OSI baseia-se no uso de identificadores de invocação, como mencionado anteriormente. No módulo *INTERFACE*, quando uma mensagem é recebida na fila externa, o seu identificador de invocação é lido e a mensagem é enviada para a instância do

módulo *CCR* adequada, através do elemento de um dos vetores de pontos de interação de índice igual ao identificador de invocação. O vetor exato a ser usado depende do tipo da mensagem recebida.

As instâncias do módulo *CCR* são criadas no momento em que o SACF envia as informações para a criação do SAO. Os passos para a criação de uma instância do CCR são descritos nos próximos parágrafos.

A variável exportada *ext_conectar* pode assumir três valores: *SEM_CONEXAO*, *A_CONECTAR* e *CONECTADO*. Inicialmente possui o valor *SEM_CONEXAO*, atribuído pelo módulo *ESPECIFICAÇÃO* (este módulo será descrito na seção seguinte). Quando uma nova instância de um módulo *CCR* deve ser criada, o módulo *INTERFACE* atribui o valor *A_CONECTAR* a esta variável e o valor do identificador de invocação da mensagem na fila à variável *ext_id_invoc*.

O módulo *ESPECIFICAÇÃO* continuamente verifica se a variável *ext_conectar* passa a assumir o valor *A_CONECTAR*. Quando isto acontece, ele cria uma nova instância do módulo *CCR* e conecta os pontos de interação desta nova instância a pontos de interação do módulo *INTERFACE*. O valor da variável *ext_id_invoc* fornece o índice dos vetores de pontos de interação do cabeçalho do módulo *INTERFACE* a que eles devem ser conectados.

Após as conexões serem feitas, o módulo *ESPECIFICAÇÃO* atribui o valor *CONECTADO* à variável *ext_conectar*. Observe que, pela regra de prioridade das transições de uma instância de módulo pai sobre as de um filho, esta transição sempre será executada após *ext_conectar* passar a ter o valor *A_CONECTAR*. A partir deste momento o módulo *INTERFACE* pode enviar a interação com a informação inicial ao CCR (título da AE remota) e atribui o valor *SEM_CONEXAO* à variável *ext_conectar*.

Para o controle sobre os processos com os quais cada invocação do módulo *CCR* se comunica o módulo *INTERFACE* mantém dois vetores, cada um de tamanho igual ao número máximo de associações permitidas no SISDI-OSI : o vetor *id_filas_usuarios*, usado para armazenar os identificadores das filas dos processos usuários do CCR, e o vetor *id_filas_SACFs*, usado para armazenar o identificador da fila do processo que implementar o SACF usado no SAO no qual a instância do CCR participa (embora, por enquanto, só haja um SACF). Estes identificadores são recebidos juntamente com o título da AE remota, enviados pelo SACF no momento de criação do SAO. Como comentado anteriormente, estes identificadores de filas só são necessários para o módulo *INTERFACE*, não sendo repassados, portanto, às instâncias do módulo *CCR*. O módulo *INTERFACE* não possui estados.

Após a criação da instância do módulo *CCR* e da conexão dos pontos de interação, para cada mensagem que for recebida para esta instância, o módulo *INTERFACE* verificará se trata-se de uma primitiva ou uma APDU e a converte em interações para serem enviadas nos canais. A conversão de primitivas implica em ler os parâmetros da primitiva através do apontador, vindo na mensagem colocada na fila (ver seção 3.5.1), preencher os parâmetros da interação, enviá-la no canal adequado, de acordo com o tipo da mensagem e o seu identificador de invocação, e desalocar os parâmetros da primitiva do *buffer* compartilhado. A conversão de APDUs consiste em apenas converter em um descritor o apontador para a área do *buffer* onde os campos da APDU estão (veja o tratamento de apontadores em ESTELLE comentado na seção 5.3.1). As APDUs são montadas no *buffer* pelas instâncias do módulo *CCR*, utilizando funções externas declaradas como PRIMITIVE. As concatenações de APDUs vindas do SACF (ou enviadas a ele) chegam como uma primitiva especial, chamada *C_PRIM_CONCAT*, cujos parâmetros são apontadores para cada uma das APDUs concatenadas (figura 3.15).

As outras transições do módulo *INTERFACE* verificam se há alguma mensagem nos canais conectados às instâncias do CCR. Pelo canal sabe-se o número de invocação. As mensagens são então convertidas e enviadas aos processos que implementam os componentes a que se destinam. Um exemplo aparece na figura 5.15, que controla o envio de uma primitiva C-BEGIN.ind de uma instância do módulo *CCR* para o seu usuário. A construção *ANY* permite que se faça uma iteração sobre o vetor *pa_USUARIO_CCR* de pontos de interação para receber interações através do canal *canal_CCR_USUARIO*, no qual a primitiva seria enviada. Se houver algum elemento deste vetor para o qual houver esta primitiva, a sua conversão será feita, utilizando-se a função *monta_buffer_C_BEGIN_ind*, que armazena os seus parâmetros no *buffer* compartilhado, retornando o endereço (na forma de um inteiro) de onde foi feito o armazenamento. Depois a mensagem é enviada ao processo usuário, utilizando-se o procedimento *envia_mensagem_fila_destino*, que possui como parâmetros o identificador da fila para a qual enviar a mensagem, o identificador de invocação da mensagem, o tipo da mensagem, primitiva ou APDU, e o endereço de onde o conteúdo da mensagem foi alocado no *buffer*. O identificador desta fila é obtido no vetor *id_fila_usuarios*.

```
TRANS
 ANY id_invoc : Tipo_Faixa_Associacoes DO
  WHEN pa_USUARIO_CCR[id_invoc].C_BEGIN_ind
   BEGIN
    apontr_info := monta_buffer_C_BEGIN_ind (id_acao_atomica,
                          id_ramo_acao_atomica,dados_usuario_op,dados_usuario);
    envia_mensagem_fila_destino (id_filas_usuarios[id_invoc],id_invoc,
                          PRIMITIVA,apontr_info);
   END;
```

Figura 5.15: Transição de conversão da interação C_BEGIN_ind

Se a mensagem tivesse que ser enviada ao SACF, o identificador da fila do processo para o qual enviar seria obtido a partir do vetor *id_filas_SACFs*.

5.3.3 O Módulo *ESPECIFICAÇÃO*

O módulo *ESPECIFICAÇÃO* (*Specification*) é o módulo mais externo de todas as especificações em ESTELLE. Na especificação para o CCR, este módulo possui a função de inicialmente criar a instância do módulo *INTERFACE* (em sua parte de "inicializações") e, depois, criar e destruir instâncias do módulo *CCR* (em sua parte de transições). Além disto atribui o valor inicial, *SEM_CONEXAO*, à variável *ext_conectar*, exportada pelo módulo *INTERFACE*. Ao módulo *ESPECIFICAÇÃO* foi atribuída a classe *SYSTEMPROCESS*, embora não haja razão especial para esta atribuição.

Este módulo possui apenas duas transições: uma que continuamente verifica se o valor da variável *ext_conectar*, exportada pelo módulo *INTERFACE*, torna-se igual a *A_CONECTAR*, como comentado na seção anterior, e outra que examina, para cada instância ativa do módulo *CCR*, se a variável exportada *ext_terminou* fica igual a *TRUE*. Quando isto acontece para uma destas instâncias, a transição a destrói. Estas transições aparecem na figura 5.16. Na primeira, a instrução

init cria a instância e a *connect* conecta os pontos de interação da instância criada aos correspondentes do módulo *INTERFACE*, de acordo com o identificador de invocação. O valor *CONECTADO* é, em seguida, atribuído à variável *ext_conectar*, exportada pelo módulo *INTERFACE*. A expressão *EXIST* e a instrução *ALL*, na segunda transição, permitem que as iterações ocorram somente sobre instâncias ativas do tipo definido na variável de iteração (*instancia_CCR*, que é uma variável de módulo). Como esta iteração é automática, as instâncias criadas não precisam ser armazenadas em um vetor.

```
{ Transicao para a criacao de uma instancia: }

TRANS
 PROVIDED (INTERFACE.ext_conectar = A_CONECTAR)
  BEGIN
    INIT inst_CCR WITH Corpo_CCR;
    CONNECT inst_CCR.pa_USUARIO TO
            INTERFACE.pa_USUARIO_CCR[INTERFACE.ext_id_invoc];
    CONNECT inst_CCR.pa_SACF_controle TO
            INTERFACE.pa_SACF_CCR_controle[INTERFACE.ext_id_invoc];
    CONNECT inst_CCR.pa_SACF_dados TO
            INTERFACE.pa_SACF_CCR_dados[INTERFACE.ext_id_invoc];
    INTERFACE.ext_conectar := CONECTADO;
  END;


  { Transicao para terminar instancias do protocolo CCR: }


TRANS
  PROVIDED EXIST instancia_CCR : Modulo_CCR
           SUCHTHAT instancia_CCR.ext_terminou = TRUE
   BEGIN
    ALL instancia_CCR : Modulo_CCR DO
     IF instancia_CCR.ext_terminou = TRUE
      THEN
       RELEASE instancia_CCR;
   END;
```

Figura 5.16: Transições do módulo *ESPECIFICAÇÃO*

5.4 A Implementação do SACF

O SACF, de acordo com o esquema geral de implementação dos processos do SISDI-OSI mostrado no capítulo 3, foi implementado em um único processo que simula internamente as suas várias

instâncias de uso nas associações. Mais especificamente, para cada uma destas instâncias de uso, as atividades exercidas pelo SACF são :

- recebe as APDUs do CCR, isoladas ou para serem concatenadas, e as mapeiam nos parâmetros de dados de usuário de primitivas da camada de Apresentação, de acordo com regras estabelecidas no documento de especificação do CCR [ISO9805], comentadas na seção 4.6.1. Em especial controla o tipo de primitiva no qual a APDU C-BEGIN-RC deve ser emitida. Esta APDU pode ser mapeada nas primitivas P-SYNC-MINOR.rsp ou P-TYPED-DATA.req, dependendo de em qual primitiva a APDU C-BEGIN-RI foi recebida (ver tabela 4.3);

- preenche os outros parâmetros das primitivas da Apresentação nas quais as APDUs são enviadas e envia as primitivas ao simulador das camadas inferiores. Estes parâmetros estão relacionados a praticamente o controle sobre pontos de sincronização e *tokens* (ver seção 4.6.1). O controle de *tokens* não foi implementado, porque estão relacionadas com o usuário do CCR. É ele quem deve garantir o atendimento aos requisitos de posse de *tokens* para a emissão de algumas primitivas, de acordo com a especificação do CCR (ver capítulo 4). O controle sobre os números dos pontos de sincronização a serem usados nas primitivas da camada de Apresentação é feito com o uso das variáveis de controle destes números especificadas em [ISO8326] e de acordo com as regras de seu uso determinadas neste documento. Os valores destas variáveis são influenciados pela emissão e recepção de primitivas de serviços desta camada que tratam de pontos de sincronização maior, menor e de ressincronização (são influenciados também pelos serviços de controle de atividades, mas estes serviços não estão sendo usados);

- recebe primitivas da camada de Apresentação (simulador), extrai as APDUs que vierem como dados de usuário e as enviam ao CCR;

- recebe as informações iniciais sobre a associação, fornecidas pelo *ESTABELECEDOR DA ASSOCIAÇÃO*. Quais são estas informações foi comentado na seção 5.1;

- informa o título da AE remota e identificadores de fila ao CCR e um identificador de fila ao ACSE. Este fornecimento de informação corresponde a uma montagem do SAO, de acordo com o contexto de aplicação que o SACF representar;

- aloca e desaloca cabeçalhos de APDUs, durante, respectivamente, o envio de APDUs para o sistema remoto e a recepção de APDUs deste sistema. Na alocação o SACF preenche os campos do cabeçalho destinados a especificar o identificador do contexto de apresentação e identificar a APDU (ver seção 3.5.2);

- trata solicitação de término de instância e repassa o pedido ao CCR (término de SAO); e

- trata indicações de erro de protocolo, enviados pelo CCR, e as repassa ao *USUÁRIO DO SAO*.

Para o SISDI-OSI foi feita uma consideração: assumiu-se que a execução das ações em uma associação e nas conexões de apresentação e de sessão que as suportam seja sincronizada. Desta maneira solicitações vindas do AP e da camada de Transporte são tratadas nestas conexões e na associação de maneira serial, como se fossem todas colocadas em uma mesma fila. Esta consideração

impede o paralelismo destro destas conexões e associação, mas ainda permite o paralelismo entre conexões e associações distintas.

Esta consideração foi feita para evitar situações de inconsistência entre os componentes do sistema, semelhantes às que foram comentadas na seção 5.3.1 sobre o fato de o CCR e seu usuário não tratarem imediatamente as mensagens enviadas de um para o outro. Para ilustrar o que ocorre com relação ao SACF, considere o exemplo da figura 5.17, em que o CCR envia uma APDU C-ROLLBACK-RI para o SACF e, logo após, a camada de Apresentação teria enviado uma primitiva P-RESYNCHRONIZE.ind para o SACF, contendo também uma APDU C-ROLLBACK-RI vinda do sistema remoto (figura 5.17a). O SACF, por ter recebido primeiro a APDU vinda do CCR, mapeiá-la-ia em uma primitiva P-RESYNCHRONIZE.req, que seria emitida à camada de Apresentação (figura 5.17b). Esta atitude provocaria uma colisão do serviço P-RESYNCHRONIZE na camada de Apresentação (figura 5.17c). Esta colisão teria que ser tratada nesta camada e poderia resultar na emissão ou não da solicitação do SACF ao sistema remoto, dependendo-se de certos fatores (qual AEI requisitou o estabelecimento da conexão de Apresentação e quais são os números dos pontos de sincronização que vão nas primitivas). Não havendo o sincronismo o SACF também teria que fazer a mesma análise para determinar como tratar a primitiva P-RESYNCHRONIZE.ind (figura 5.17c). Se a colisão foi resolvida em favor do sistema local, a primitiva P-RESYNCHRONIZE.ind teria que ser descartada, porque, em caso contrário, o CCR receberia a APDU C-ROLLBACK-RI, que causaria a emissão, posteriormente, da APDU C-ROLLBACK-RC para o sistema remoto. Com isto o sistema remoto receberia a APDU C-ROLLBACK-RI seguida da APDU C-ROLLBACK-RC.

Outras situações semelhantes podem ocorrer. Estas situações acabam dificultando a implementação dos componentes, duplicando-se tarefas e aumentando a complexidade das máquinas de estado dos protocolos. A consideração de sincronismo facilita a implementação, permitindo-se uma garantia maior do correto funcionamento do sistema.

A implementação do SACF segue os mesmos princípios da implementação do CCR. A estrutura em módulos e os canais usados são mostrados na figura 5.18. Nesta figura os módulos *APRESENTAÇÃO*, *CCR*, *USUÁRIO DO SAO* e *ESTABELECEDOR DA ASSOCIAÇÃO* são representados pelo módulo *INTERFACE*, que simula a interação do SACF com estes módulos, que aparecem, por isto, tracejados nesta figura. Os canais portanto conectam, na implementação, pontos de interação de instâncias do módulo *SACF* a pontos de interação do módulo *INTERFACE*, de modo análogo ao feito na implementação do CCR.

Só há uma instância do módulo *INTERFACE*, mas uma instância do módulo *SACF* para cada associação em que este SACF for usado. Os módulos *INTERFACE* e *ESPECIFICAÇÃO* são análogos aos de mesmo nome do *CCR* (seção 5.3).

O cabeçalho do módulo *SACF* aparece na figura 5.19. Nele há um ponto de interação para cada módulo com o qual o SACF interage segundo a figura 5.1: há um ponto para o módulo *ESTABELECEDOR DE ASSOCIAÇÃO* (*pa_ESTABELECEDOR_ASSOC*), um para o módulo usuário do SAO (*pa_USUARIO*), dois para o CCR, sendo um para troca de dados e outra para troca de primitivas de controle (respectivamente, *pa_CCR_dados* e *pa_CCR_controle*) e um para a camada de Apresentação (*pa_APRESENTACAO*). No cabeçalho há ainda uma variável exportada, *ext_terminou*, usada para terminar a instância do módulo *SACF*, como na implementação do CCR. O módulo *INTERFACE* terá, portanto, vetores de pontos de interação para receber e emitir as interações para cada instância do módulo *SACF*.

As declarações dos canais usados pelo módulo *SACF* são as das figuras 5.20 e 5.21.

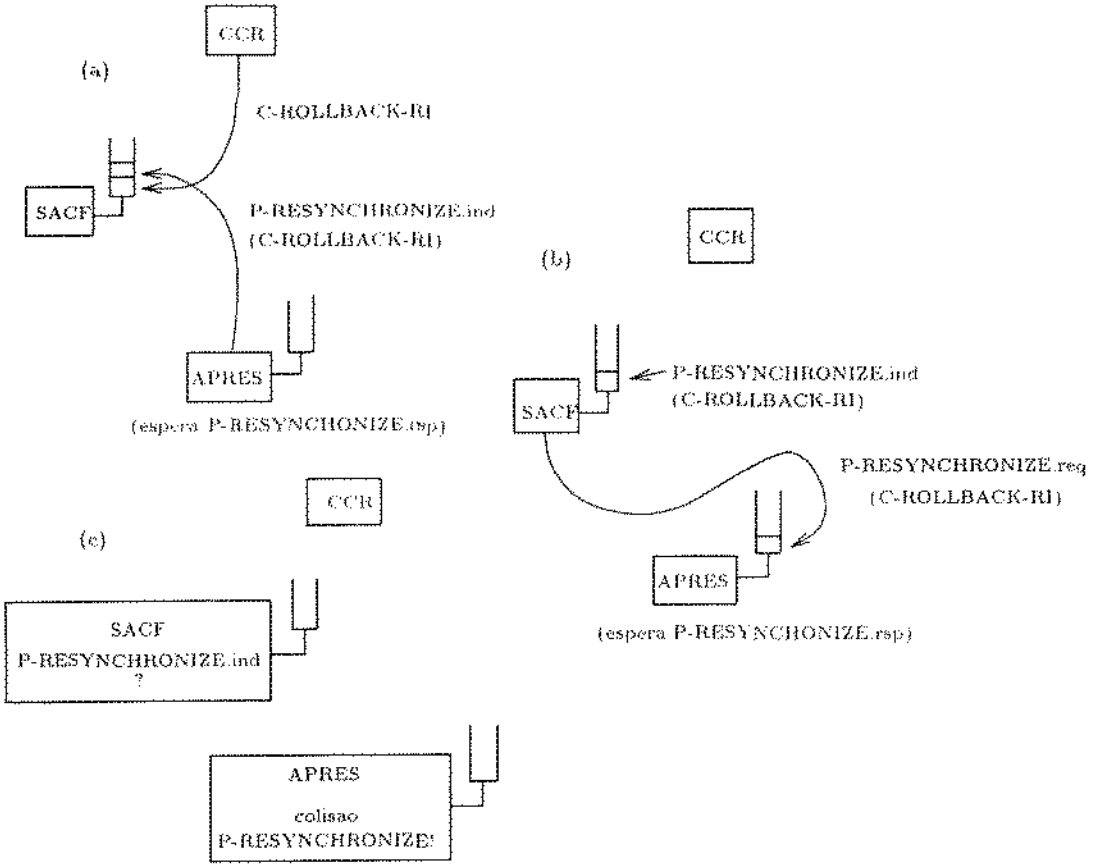

Figura 5.17: Exemplo de problema causado por falta de sincronismo entre componentes

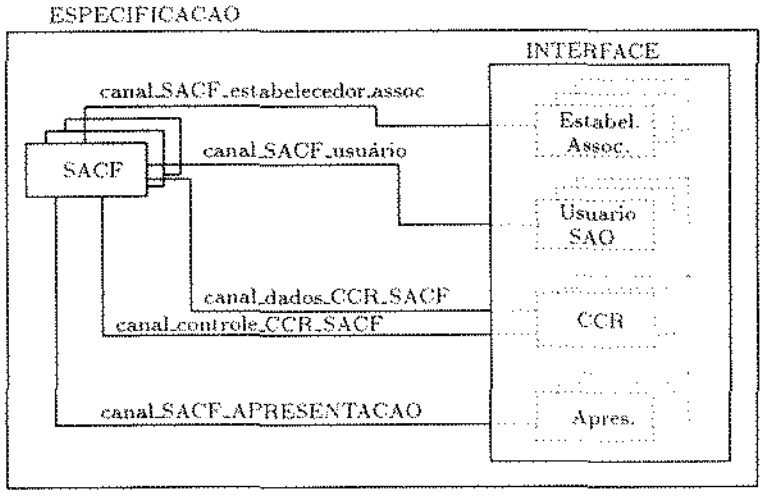

Figura 5.18: Estrutura de implementação do SACF

```
MODULE Modulo_SACF PROCESS;
  IP
    pa_ESTABELECEDOR_ASSOC : canal_SACF_ESTABELECEDOR_ASSOC (SACF) COMMON QUEUE;
    pa_USUARIO : canal_SACF_USUARIO (SACF) COMMMON QUEUE;
    pa_CCR_dados : canal_dados_CCR_SACF (SACF) COMMON QUEUE;
    pa_CCR_controle : canal_controle_CCR_SACF (SACF) COMMON QUEUE;
    pa_APRESENTACAO : canal_SACF_APRESENTACAO (SACF) COMMON QUEUE;

  EXPORT
    ext_terminou : BOOLEAN;
END;
```

Figura 5.19: Cabeçalho do módulo *SACF*

```
CHANNEL Canal_SACF_ESTABELECEDOR_ASSOC (ESTABELECEDOR,SACF);
  BY ESTABELECEDOR:
   S_INFO_INICIAIS ( titulo_AE_remota : Tipo_Titulo_AE;
                     numero_ponto_sincr_inicial : INTEGER;
                     atrib_inicial_tokens : Tipo_Tokens );

CHANNEL Canal_SACF_USUARIO (USUARIO,SACF);
  BY USUARIO:
    S_TERMINA_INVOCACAO;
  BY SACF:
    S_SAO_MONTADO;
    S_ERRO_DE_PROTOCOLO_LOCAL ( componente : Tipo_Componentes;
                                prim : INTEGER );
    S_ERRO_DE_PROTOCOLO_REMOTO ( componente : Tipo_Componentes;
                                 apdu : INTEGER );

CHANNEL canal_controle_CCR_SACF (SACF, CCR) ;

     ...

CHANNEL canal_dados_CCR_SACF (SACF, CCR);

     ...
```

Figura 5.20: Canais usados pelo SACF para comunicação com o *ESTABELECEDOR DE ASSOCIAÇÃO*, com seu usuário e com o CCR

```
CHANNEL canal_SACF_APRESENTACAO (SACF,APRESENTACAO);
  BY SACF:
    P_TYPED_DATA_req ( dados_usuario : Tipo_Dados_Usuario );
    P_SYNC_MINOR_req ( tipo : Tipo_Obrigatorio_ou_Opcional;
                       numero_ponto_serial_sincr : INTEGER;
                       dados_usuario_op : Presente_ou_Ausente;
                       dados_usuario : Tipo_Dados_Usuario  );
    P_SYNC_MINOR_rsp ( numero_ponto_serial_sincr : INTEGER;
                       dados_usuario_op : Presente_ou_Ausente;
                       dados_usuario : Tipo_Dados_Usuario  );
    P_SYNC_MAJOR_req ( ... );
    P_SYNC_MAJOR_rsp ( ... );
    P_RESYNCHRONIZE_req ( ... );
    P_RESYNCHRONIZE_rsp ( ... );
  BY APRESENTACAO :
    P_TYPED_DATA_ind ( dados_usuario : Tipo_Dados_Usuario );
    P_SYNC_MINOR_ind ( tipo : Tipo_Obrigatorio_ou_Opcional;
                       numero_ponto_serial_sincr : INTEGER;
                       dados_usuario_op : Presente_ou_Ausente;
                       dados_usuario : Tipo_Dados_Usuario  );
    P_SYNC_MINOR_cnf ( ... );
    P_SYNC_MAJOR_ind ( ... );
    P_SYNC_MAJOR_cnf ( ... );
    P_RESYNCHRONIZE_ind ( ... );
    P_RESYNCHRONIZE_cnf ( ... );
```

Figura 5.21: Canal usado pelo SACF para comunicação com a camada de Apresentação

O canal *canal_SACF_ESTABELECEDOR_ASSOC* serve para que o componente *ESTABELECEDOR DA ASSOCIAÇÃO* passe as informações sobre o estabelecimento da associação. Como já comentado, fornece o título da AE remota, para ser repassada ao CCR, o número inicial do ponto de sincronização combinado, e a posse inicial de tokens (embora esta informação não seja usada por enquanto). Observe que o identificador da fila do usuário do SAO não aparece na interação, pois é tratado apenas pelo módulo *INTERFACE*, como no caso do CCR. O canal *canal_SACF_USUARIO* serve para o SACF se comunicar com o *USUÁRIO DO SAO* e especifica que o SACF pode receber um pedido de término da invocação (*S_TERMINA_INVOCACAO*) e pode enviar indicações de erros de protocolo (*S_ERRO_DE_PROTOCOLO_LOCAL* e *S_ERRO_DE_PROTOCOLO_REMOTO*), além da indicação de término de montagem do SAO (*S_SAO_MONTADO*). Para a comunicação com o CCR há dois canais : *canal_controle_CCR_SACF* e *canal_dados_CCR_SACF*. Estes canais são os mesmos canais de mesmo nome que aparecem na especificação do CCR, que servem, respectivamente, para a troca de primitivas de controle e APDUs. Não são, portanto, repetidos na figura. O canal *canal_SACF_Apresentacao* é usado para a troca de primitivas com a camada de Apresentação (simulador). Este canal contém todas as primitivas de serviços da Apresentação usadas para a transmissão de APDUs do CCR. Em algumas interações foram usadas reticências, por economia de espaço. Os parâmetros das interações são exatamente os mesmos das primitivas especificadas em [CCITT216], mas tendo os parâmetros opcionais tratados da mesma maneira que os das primitivas do CCR (ver seção 5.3.1).

O módulo *INTERFACE* para a especificação do SACF trata de alguns identificadores de filas, de modo análogo ao caso para o CCR. Quando as informações iniciais sobre a associação são enviadas, é enviado também o identificador da fila do usuário do SAO, que, por enquanto, é sempre o identificador da fila do processo de recebimento de mensagens do componente USUÁRIO. Quando a interação *C_TITULO_AE_REMOTA* é enviada ao CCR, o módulo *INTERFACE* insere o identificador de fila do processo que implementa o SACF e o identificador de fila do processo usuário do CCR, que, por enquanto, é também o identificador do processo de recepção de mensagens do USUÁRIO.

A transição de inicialização do módulo *SACF* apenas inicializa a variável *ext_terminou* com o valor *FALSE*. O módulo *SACF* não possui estados.

As transições do *SACF* executam as funções mostradas no início desta seção. O conjunto de transições é composto de:

1. uma transição para tratar a chegada das informações iniciais sobre a associação estabelecida, enviadas pelo módulo *ESTABELECEDOR DE ASSOCIAÇÃO*. Esta transição aparece na figura 5.22, e trata a interação *S_INFO_INICIAIS*, que chega pelo ponto de interação *pa_ESTABELECEDOR_ASSOC*. Os parâmetros desta interação aparecem na figura 5.20.

   O tratamento desta interação é "inicializar" variáveis locais com as informações recebidas, informar ao CCR o título da AE remota (como parte da montagem do SAO) e indicar ao seu usuário de que o SAO está montado. O título da AE remota (*titulo_AE_remota*) e a atribuição inicial de tokens (*atrib_inicial_tokens*) são apenas armazenados em variáveis locais (*titulo_AE_estabelecedora_assoc* e *posse_tokens*). O número do ponto de sincronização inicial (*numero_ponto_sincr_inicial*) serve para iniciar as variáveis *V_M* e *V_A*, para controle do número seqüencial dos pontos de sincronização. As outras variáveis para este fim, *V_R* e *Vsc* são inicializadas com os valores mostrados. Os valores dados a estas variáveis devem estar de acordo com o especificado em [ISO8326]:

```
{ Tratamento de S_INFO_INICIAIS : }

 WHEN pa_ESTABELECEDOR_ASSOC.S_INFO_INICIAIS
  BEGIN
   titulo_AE_estabelecedora_assoc := titulo_AE_remota;
   V_M := numero_ponto_sincr_inicial;
   V_A := numero_ponto_sincr_inicial;
   V_R := 0;
   Vsc := FALSE;
   posse_tokens := atrib_inicial_tokens;
   OUTPUT pa_CCR_controle.C_TITULO_AE_REMOTA(titulo_AE_remota);
   OUTPUT pa_USUARIO.S_SAO_MONTADO;
  END;
```

Figura 5.22: Transição para tratamento das informações iniciais

2. uma transição para controlar o pedido de término da instância, ou seja, tratar a mensagem *S_TERMINA_INVOC* no ponto de interação *pa_USUARIO*. Esta transição, mostrada na figura 5.23, atribui o valor *TRUE* à variável *ext_terminou*, exatamente do mesmo modo e com o mesmo objetivo da transição que trata término de invocação do módulo *CCR*, mas também envia uma interação de término de invocação para cada ASE, com exceção do ACSE, que fizer parte do SAO (no caso, somente o CCR, enviando uma interação *C_TERMINA_INVOCACAO*);

```
{ Tratamento de S_TERMINA_INVOCACAO : }

 WHEN pa_USUARIO.S_TERMINA_INVOCACAO
  BEGIN
   OUTPUT pa_CCR_controle.C_TERMINA_INVOCACAO;
   ext_terminou := TRUE;
  END;
```

Figura 5.23: Transição para tratamento da solicitação de término do SAO

3. uma transição para cada envio de APDU feito pelo CCR. Este envio pode ser de ou uma APDU isolada ou de duas, para concatenação. Esta transição trata de enviá-las para o sistema remoto. Mapeia-a(s) no campo de dados do usuário de primitivas da Apresentação, monta o cabeçalho para cada APDU, de acordo com o comentado na seção 3.5.2, preenche os outros parâmetros da primitiva na qual a(s) APDU(s) será(ão) enviada(s) e controla, se for o caso, as variáveis associadas aos números dos pontos de sincronização definidas na camada de Sessão [ISO8326]. Estas variáveis são usadas pelo SACF para que ele possa controlar os valores relativos a estes pontos que serão atribuídos a parâmetros de primitivas de serviços da Apresentação. Um exemplo de transição para o tratamento de uma APDU que não será

concatenada aparece na figura 5.24.

```
{ Tratamento de C-BEGIN-RI : }

  WHEN pa_CCR_dados.C_BEGIN_RI
   BEGIN

    { Montagem do cabecalho : }

     dados_usuario.apontr_info := monta_cabecalho_apdu_ccr(apontr_info);
     dados_usuario.apontr_prox := DESCRITOR_NULO;

    { Envio da primitiva aa Apresentacao : }

     OUTPUT pa_APRESENTACAO.P_SYNC_MINOR_req (Opcional,V_M,
                                              PRESENTE,dados_usuario);

    { Armazenamento do numero do ponto de sincronizacao associado a
      C-BEGIN-RI e a posse de tokens neste ponto : }

     numero_ponto_C_BEGIN := V_M;
     posse_tokens_C_BEGIN := posse_tokens;

    { Tratamento das variaveis de controle dos numeros de pontos de
      sincronizacao : }

     IF (Vsc = TRUE) THEN
      BEGIN
       V_A := V_M;
       Vsc := FALSE;
      END;
     V_M := V_M + 1;
  END;
```

Figura 5.24: Exemplo de controle sobre recursos das camadas inferiores

Nesta transição primeiramente se cria o cabeçalho da APDU. Em seguida os parâmetros da primitiva na qual a APDU será enviada são preenchidos de acordo com as regras estabelecidas no documento do CCR [ISO9805] e a primitiva é emitida. Como estes parâmetros são preenchidos foi mostrado na seção 4.6.1. Depois o número do ponto de sincronização associado ao envio da APDU C-BEGIN-RI é armazenado em uma variável, para o caso de haver ROLLBACK, quando haveria ressincronização para este ponto (início do ramo da ação atômica). A posse dos *tokens* também é armazenada, para ser restaurada no caso de ressincronização. No final as operações que devem ser feitas nas variáveis de controle associadas aos pontos de

sincronização são feitas (de acordo com [ISO8326]).

No caso de haver concatenação de APDUs, a transição invocaria rotinas que montam a estrutura de concatenação de acordo com os tipos dos campos de dados de usuário de primitivas da Apresentação, mostrados na seção 3.5.2.

4. uma transição para tratar a chegada de cada primitiva. Nesta transição as APDUs são extraídas e enviadas ao CCR. Novamente as variáveis de controle de pontos de interação são tratadas, dependendo-se do tipo da primitiva. Um exemplo destas transições pode ser o da primitiva P_TYPED_DATA.ind, na qual vários tipos de APDUs podem vir. Este exemplo aparece na figura 5.25. A transição desta figura desaloca o cabeçalho da APDU (através da função externa *desaloca_cabecalho*, que retorna um apontador para a APDU em si), determina o tipo da APDU que veio na primitiva (através da função externa *obtem_tipo_apdu*) e a envia ao CCR. Nenhuma das concatenações feitas no SISDI-OSI é mapeada nas primitivas do serviço P-TYPED-DATA;

```
{ Tratamento de P-TYPED-DATA.ind : }

WHEN pa_APRESENTACAO.P_TYPED_DATA_ind
 BEGIN
  apontr_apdu := desaloca_cabecalho (dados_usuario.apontr_info);
  tipo_apdu := obtem_tipo_apdu (apontr_apdu);
  CASE (tipo_apdu) OF
   C_BEGIN_RC : OUTPUT pa_CCR_dados.C_BEGIN_RC (apontr_apdu);
   C_PREPARE_RI : OUTPUT pa_CCR_dados.C_PREPARE_RI (apontr_apdu);
   C_READY_RI : OUTPUT pa_CCR_dados.C_READY_RI (apontr_apdu);
   C_RECOVER_RI : OUTPUT pa_CCR_dados.C_RECOVER_RI (apontr_apdu);
   C_RECOVER_RC : OUTPUT pa_CCR_dados.C_RECOVER_RC (apontr_apdu);
  END;  { CASE }
 END;
```

Figura 5.25: Transição de tratamento de P-TYPED-DATA.ind

5. duas transições para o tratamento de indicações de erro de protocolo vindas do CCR : uma para tratar a indicação de erro de protocolo causado por emissão de primitiva (erro local) e a outra para tratar a indicação de erro de protocolo causado pela chegada de uma APDU (erro remoto) (figura 5.26). O SACF avisa o erro, o evento que o gerou e em qual componente (por enquanto, só o CCR) a seu usuário e termina a sua invocação, através da atribuição do valor *TRUE* à variável *ext_terminou*.

5.5 Implementação da Simulação das Camadas Inferiores

No SISDI-OSI as implementações de todas as unidades funcionais das camadas de Apresentação e Sessão estão sendo feitas [Amb], como também uma interface que mapeia os serviços de Transporte

```
{ Tratamento de ERRO DE PROTOCOLO LOCAL do CCR : }

  WHEN pa_CCR_controle.C_ERRO_DE_PROTOCOLO_LOCAL
   BEGIN
    OUTPUT pa_USUARIO.S_ERRO_DE_PROTOCOLO_LOCAL (CCR,prim);
    ext_terminou := TRUE;
   END;

{ Tratamento de ERRO DE PROTOCOLO REMOTO do CCR : }

  WHEN pa_CCR_controle.C_ERRO_DE_PROTOCOLO_REMOTO
   BEGIN
    OUTPUT pa_USUARIO.S_ERRO_DE_PROTOCOLO_REMOTO (CCR,apdu);
    ext_terminou := TRUE;
   END;
```

Figura 5.26: Transições de tratamento de indicações de erro de protocolo feitas pelo CCR

classe 4 no protocolo TCP/IP [Saka], de maneira que no futuro duas máquinas distintas poderão se comunicar utilizando o SISDI-OSI sobre uma rede TCP/IP.

Para a implementação do CCR foi usada uma simulação das camadas inferiores. As implementações do SISDI-MAP para as camadas de Apresentação e Sessão ainda não foram convertidas para o SISDI-OSI e ainda não estão implementados certos serviços utilizados pelo CCR. Esta simulação foi baseada em uma outra, feita para o SISDI-MAP [Jac89], quando este ainda não dispunha nem das implementações citadas acima. Esta nova simulação, no entanto, foi totalmente reimplementada, devido às características do novo ambiente. A simulação que havia no SISDI-MAP tratava condições de erro. Esta nova versão não trata.

Esta simulação das camadas inferiores é bastante simples, feita apenas para permitir o teste dos protocolos de Aplicação, através da comunicação entre duas instâncias do protocolo, mas na mesma máquina. O que esta simulação faz é receber as primitivas de serviços da camada de Apresentação do tipo *request* e *response* emitidas pela camada de Aplicação com um identificador de invocação e gerar as primitivas correspondentes dos tipos *indication* e *confirm*, emitindo-as de volta para a camada de Aplicação, com um outro identificador de invocação. Haverá invocações independentes dos protocolos da Aplicação para simular cada um dos sistemas que se comunicam através de uma associação. Este módulo de simulação faz o mapeamento entre estas invocações, estabelecendo, com isto, a associação hipotética, mas utilizando, do ponto de vista dos protocolos de Aplicação, duas associações. A figura 5.2 ilustra o uso deste simulador. Observe que do ponto de vista da Aplicação há duas conexões de apresentação. O simulador fará, portanto, um mapeamento entre estas duas conexões.

A simulação é implementada como um processo UNIX independente e possui também uma única fila externa para a recepção das mensagens. O código simplesmente fica em um *loop* eterno, que espera uma primitiva da camada de Apresentação chegar na fila, trata-a e passa a esperar uma nova primitiva.

Todas as primitivas emitidas em uma mesma associação e na sua conexão de apresentação terão o mesmo identificador de invocação. A cada conexão de apresentação corresponderá um identificador de invocação diferente. A simulação terá que simplesmente manter o par de identificadores das invocações que devem ser mapeados e o identificador de fila do processo que deve receber as mensagens.

A estrutura de dados necessária para esta simulação é um vetor, chamado *con_tab*, de tamanho igual ao número máximo de associações que o sistema permitir, onde cada componente é um registro do tipo *Tipo_Info_Mapeamento_Conexoes*. A definição de *Tipo_Info_Mapeamento_Conexoes* e a declaração de *con_tab* aparecem na figura 5.27.

```
typedef struct
        {
          Tipo_Id_Fila id_fila_usuario;
          Tipo_Id_Invoc id_invoc_par;
        } Tipo_Info_Mapeamento_Conexoes;

Tipo_Info_Mapeamento_Conexoes con_tab [NUMMAX_ASSOCIACOES];
```

Figura 5.27: Estruturas de dados mantidas pelo simulador das camadas inferiores

No vetor *con_tab* o seu elemento de índice *i* sempre armazenará as informações para a conexão de apresentação associada ao identificador de invocação *i*. Para uma conexão, o campo *id_fila_usuario* é usado para armazenar o identificador da fila do processo para o qual a primitiva gerada na conexão par deve ser enviada e o campo *id_invoc_par* é usado para armazenar o identificador da conexão par.

O mapeamento é estabelecido durante o tratamento das primitivas do serviço P-CONNECT, que é o que requisita o estabelecimento das conexões de apresentação. Quando uma primitiva P-CONNECT.req é enviada ao simulador, este obtém um identificador de invocação ainda não usado. Os elementos do vetor *con_tab* de índices iguais a este identificador e ao identificador de invocação da primitiva P-CONNECT.req serão usados para mapear as duas conexões. O campo *id_fila_usuario* de ambos os elementos do vetor terão o mesmo valor, que, por enquanto, é o identificador da fila do SACF.

Feito este mapeamento, para cada primitiva que chega à simulação, que é do tipo *request* ou *response*, o tratamento que se faz é o seguinte:

- se a primitiva correspondente, do tipo *indication* ou *confirm*, possui exatamente os mesmos parâmetros da primitiva recebida, o simulador apenas troca o identificador do tipo da primitiva no *buffer* e envia uma mensagem para a camada de Aplicação com o identificador da invocação no sistema remoto simulado. As informações sobre qual é o identificador de invocação e da fila são encontrados no vetor *con_tab*. A figura 5.28 ilustra esta situação para o caso das primitivas de serviço P-TYPED-DATA;

- se, por outro lado, não houver um mapeamento direto entre os parâmetros das primitivas, a nova primitiva é montada no *buffer* com os valores dos parâmetros obtidos a partir da primitiva recebida da Aplicação e de valores que seriam determinados pela própria camada

de Apresentação. O identificador da fila destino e o identificador de invocação com que a primitiva deve ser enviada são obtidos em *con_tab*. A primitiva recebida é então desalocada do *buffer*. A figura 5.29 ilustra esta situação para o caso das primitivas P-RESYNCHRONIZE.req e P-RESYNCHRONIZE.ind.

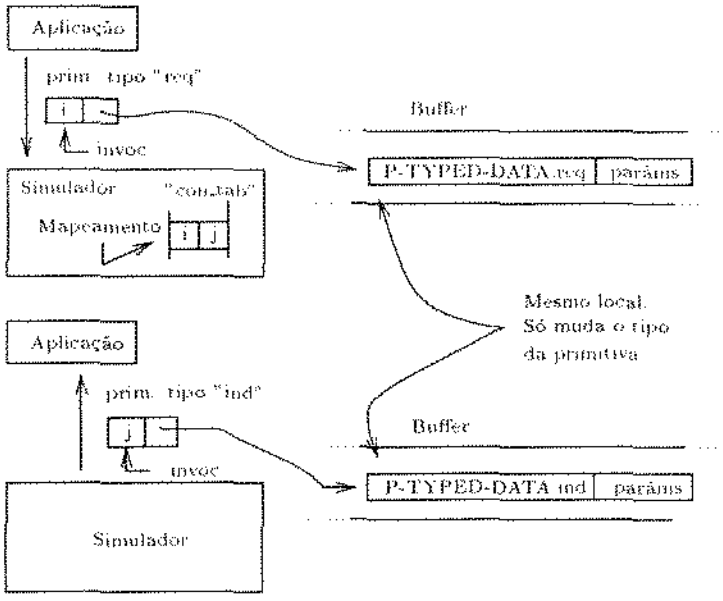

Figura 5.28: Tratamento de primitivas com mesmos parâmetros

As primitivas para estabelecimento e liberação das conexões de Apresentação seguem este mesmo tratamento, exceto que são sempre enviadas ao ACSE.

5.6 Testes das Implementações

Os testes dos componentes foram feitos por partes. Primeiramente foram testadas as funções de suporte, ou seja, de manipulação de filas, do *buffer* e do semáforo. Para estes testes foram criados pequenos programas que utilizam o recurso em questão. Depois os protocolos foram testados.

As filas foram testadas com pequenos programas que enviam mensagens uns aos outros. Para as rotinas do *buffer*, por exemplo, foi criado um programa que aloca e desaloca áreas do *buffer*, a partir de solicitações feitas pelo usuário da máquina. Pode-se especificar o tamanho desejado para a área alocada e qual área específica se quer desalocar. Este programa permite também que o usuário veja um esquema da ocupação do *buffer*. Este esquema é fornecido textualmente.

Para o teste das rotinas de semáforo, foi feito um pequeno programa que permite a seu usuário requerer o semáforo, liberá-lo e imprimir o valor corrente do semáforo. Invocando este programa várias vezes, podem-se simular situações de concorrência.

Para a implementação do CCR foram criadas seqüências de testes. Nestas seqüências foram reproduzidas seqüências corretas de efetivação, de *rollback* e erros de protocolos, tanto locais, quanto remotos. Para a criação destes erros remotos, foi criado um processo que consiste do próprio CCR,

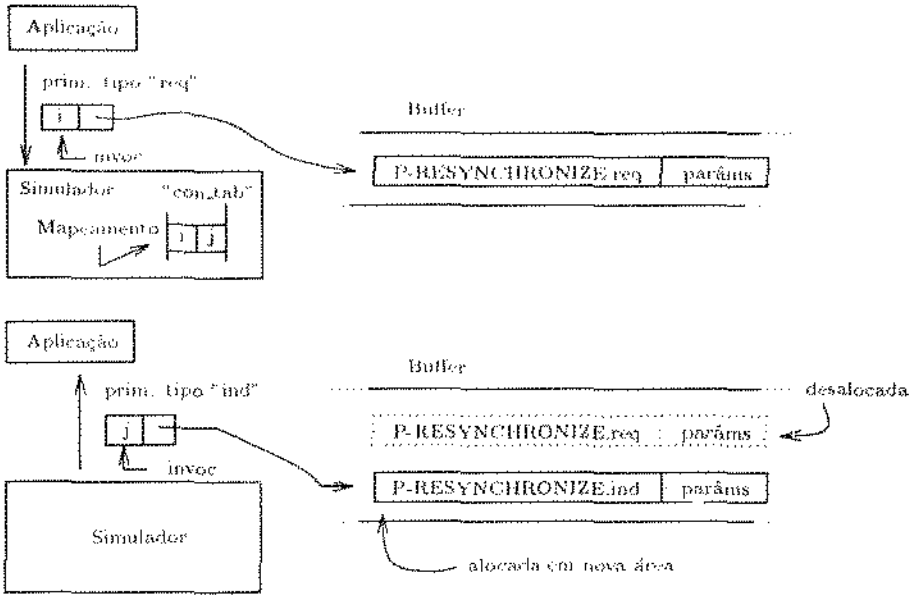

Figura 5.29: Tratamento de primitivas com parâmetros diferentes

mas sem qualquer controle de seqüenciamento de eventos. Portanto este processo simplesmente gera APDUs, a partir da recepção de primitivas de serviços, e primitivas, a partir da chegada de APDUs. O SACF e o processo que implementa o USUÁRIO foram alterados para, em uma associação específica (um valor específico para o identificador de invocação), enviar os eventos a este processo ao invés de ao processo que implementa o CCR completo. Fazendo-se uma comunicação entre a implementação completa do CCR e a que não contém controle de sequênciamento, podem-se testar não só situações de erro como também situações de colisões.

Para se testar o SACF, como a implementação deste componente não possui estados, foram emitidas APDUs do CCR e verificou-se as primitivas que eram emitidas ao simulador e qual era a APDU (ou APDUs) que o SACF emitia na outra associação. Para a emissão das APDUs utilizou-se a implementação do CCR sem controle de seqüenciamento. O controle que o SACF faz de pontos de sincronização foram observados imprimindo-se os valores das váriaveis de controle do número destes pontos.

O ACSE já existia implementado no SISDI-MAP e foi apenas migrado para o SISDI-OSI. No entanto somente foram usadas suas primitivas para estabelecimento de associação, únicas necessários à implementação do CCR. Estas primitivas foram usadas várias vezes durante os testes do CCR. Testes para verificar a completa correção da implementação deste protocolo precisam ser ainda feitas.

A implementação da simulação das camadas inferiores é simples. O seu teste foi feito enviando primitivas da camada de Apresentação e recebendo as primitivas retornadas pelo simulador. Este componente não tem estados.

O fato de se assumir o sincronismo comentado na seção 5.4 facilita bastante os testes, pois não há paralelismo em uma associação e nas conexões de apresentação e sessão que a suportam. Além dos testes para uma só associação, foram criadas situações com mais de uma associação.

Capítulo 6

Conclusão

Os objetivos desta dissertação foram a definição mais precisa de funcionalidades para a camada de Aplicação do modelo RM-OSI/ISO, descrever a estrutura geral do SISDI-OSI, nome da nova versão do sistema SISDI-MAP para o ambiente UNIX, e a implementação do protocolo CCR neste sistema.

Esta definição de funcionalidades constitui-se de um refinamento dos conceitos presentes no documento da ISO que descreve a estrutura para a camada de Aplicação, [ISO9545]. Os conceitos presentes neste documento foram analisados e procurou-se explicá-los mais claramente e especificar as funcionalidades relacionadas a estes conceitos que aparecem nos protocolos desta camada.

Nesta análise foram comentados aspectos sobre os relacionamentos entre as associações de uma AEI (*Application Entity Invocation*), o procedimento de estabelecimento de associação, endereçamento, o relacionamento do protocolo DS com estes procedimentos, o controle feito pela camada de Aplicação sobre os recursos das camadas inferiores, além da definição de funcionalidades mais precisas para as funções de controle MACF (*Multiple Association Control Function*) e SACF (*Single Association Control Function*). O modelo proposto é descrito no capítulo 2 e em [Sil93].

Na análise das funcionalidades presentes nos protocolos procurou-se extrair apenas funcionalidades que fossem genéricas, não específicas de um protocolo em particular, que poderiam ser usadas para vários contextos de aplicação. Este modelo pretende servir de uma descrição geral sobre o funcionamento da camada de Aplicação e formar uma base conceitual, da qual seja mais fácil derivar estruturas para implementação do que se partir da descrição de estrutura feita pela ISO. Nestas estruturas para implementação seriam descritos modos específicos de se implementar os componentes apresentados neste modelo e seus relacionamentos. Um esquema de implementação para os protocolos ACSE, MMS e DS aparece em [Par90], [Chi90] e [Coo90].

O SISDI-OSI trata-se de um sistema de comunicação didático, que, por isto, foi estruturado mais em função de simplicidade do que de eficiência. Mas a construção deste sistema é útil para que se verifiquem novos relacionamentos ou restrições nos relacionamentos que há entre os componentes da camada de Aplicação, os tipos mais precisos dos dados e o fluxo destes entre os componentes desta camada. O sistema é simples, com certas restrições, como só permitir certas concatenações. No entanto ele procura permitir fáceis alterações para que mais funcionalidades sejam acrescentadas.

Neste sistema cada ASE é implementado como um processo UNIX independente. A cada contexto de aplicação suportado no sistema haveria também um destes processos para o código do SACF deste contexto. O componente *ESTABELECEDOR DE ASSOCIAÇÃO* determinaria,

dentre os processos que implementam SACFs, aquele que seria usado em uma associação, de acordo com o contexto de aplicação negociado.

Por enquanto só foi implementado um SACF, que contém as funções atribuídas a SACFs, de acordo com o modelo proposto para a camada, relacionadas apenas ao *uso direto* do CCR. Este uso é apresentado no documento de especificação do protocolo CCR. O SACF, com isto, ficou responsável por receber APDUs emitidas pelo CCR, concatená-las ou não, dependendo do caso e mapeá-las em parâmetros de dados do usuário de primitivas de serviços da camada de Apresentação. Na recepção de primitivas vindas desta camada, realiza o processo inverso. Quanto ao tratamento de recursos das camadas inferiores, está tratando apenas os números que são associados a pontos de sincronização. O controle sobre *tokens* não foi feito, pois é bastante dependente do protocolo TP, que será o usuário do CCR no SISDI-OSI. Na implementação o SACF trata ainda pedidos de término do SAO e indicações de erro de protocolo. Ele transforma o pedido de término do SAO em solicitações de término de instâncias de ASEs (só do CCR, por enquanto).

O CCR foi implementado desempenhando apenas suas funções que não dependem do contexto de aplicação usado. Desta maneira a implementação fica modular, no sentido de que foram colocadas no SACF as funcionalidades dependentes do contexto de aplicação escolhido. Quais primitivas devem ser usadas para se mapear uma APDU ou quais concatenações podem ser feitas (embora no SISDI-OSI somente concatenações indicadas pelo CCR estão sendo feitas) variam de contexto de aplicação para contexto de aplicação, pois podem depender das regras de outros ASEs que fizerem parte do SAO. A especificação das funcionalidades contribuiu pelo fato de o documento do CCR não fazer distinção entre qual componente da camada deve exercer qual função. Foi útil, portanto, para compreender o CCR em função dos relacionamentos que tem com outros componentes da camada de Aplicação. O modelo procura mostrar relacionamentos que não aparecem explícitos nos documentos.

A implementação também mostrou a troca de informações iniciais para a montagem do SAO. A partir de informações iniciais sobre a associação estabelecida, passadas pelo *ESTABELECEDOR DE ASSOCIAÇÃO*, o SACF escolhido para o SAO enviará informações para cada ASE, de acordo com o seu contexto de aplicação. Desta maneira os ASEs ficam parametrizados, recebendo do SACF as informações necessárias para o seu uso em um SAO.

Observou-se que o SACF possui muitas funcionalidades e trata informações sobre vários componentes. Vai representar, provavelmente, o componente mais complexo em implementações da camada de Aplicação.

A implementação do CCR, no entanto, é ainda bastante simples para se obter resultados práticos sobre a estrutura proposta. Outras implementações em especial, a implementação do protocolo TP, trarão mais informações. Esta implementação faz parte de uma outra tese de mestrado, em andamento [Fuj].

Na conversão do sistema para o ambiente UNIX foi experimentada a inclusão do compilador ASN.1 apresentado em [Res92] e definiu-se a maneira como as estruturas geradas por ele poderiam ser utilizadas. A estrutura para as APDUs influenciam na estrutura para as primitivas de serviços, que possuem vários parâmetros correspondentes a campos de APDUs, e as decisões sobre quais informações seriam colocadas nas mensagens que vão nas filas dos processos e quais ficariam no *buffer*. No esquema escolhido identifica-se na mensagem colocada na fila apenas o tipo da mensagem e o local onde ela está no *buffer*. Na área compartilhada fica realmente o conteúdo da mensagem. Este esquema evita cópias de dados e permite, através de um campo inicial na área do *buffer* onde estão as APDUs e primitivas alocadas, uma rápida conversão entre primitivas e APDUs ainda não

codificadas, evitando cópias quando a primitiva e a APDU gerada forem iguais.

As funções de codificação e decodificação são usadas na camada de Apresentação. No sentido de transmissão de dados, a rotina de codificação é chamada para codificar a PDU de Apresentação e as APDUs que estiverem como seus dados de usuário. No sentido de recepção, a rotina de decodificação monta as estruturas em C a partir das PDUs codificadas. O SACF se encarrega de enviar as APDUs para quem deve tratá-las.

O uso dos vários apontadores na área compartilhada, derivados dos tipos em C gerados pelo compilador ASN.1, permitiram um esquema bastante flexível para a concatenação de APDUs. Cada protocolo apenas monta uma espécie de cabeçalho para a APDU, dependendo do tipo de primitiva na qual a APDU vai como dado de usuário, e o SACF se encarrega de concatenar as APDUs segundo suas próprias regras, manipulando apenas apontadores. Na dissertação mostrou-se qual componente trata cada estrutura de dado que monta as APDUs.

O uso destes apontadores, no entanto, criou a necessidade de se usarem funções genéricas para a desalocação e cópia da árvore de APDUs que pode surgir. O encapsulamento de APDUs, umas dentro das outras, faz com que estas rotinas tenham que ir percorrendo a árvore, identificando o tipo de cada APDU e invocando rotinas específicas para a cópia ou desalocação de cada uma delas. A identificação é possível através de campos existentes nas próprias estruturas geradas pelo compilador.

Para a implementação foi suposta uma sincronização entre os eventos em uma associação e na conexão de apresentação e sessão que a suportam. Esta suposição foi bastante útil, pois evitou complicações na implementação dos componentes, principalmente do SACF, para tratar das inconsistências devido à execução em paralelo das atividades exercidas nestas conexões e associação. Esta suposição faz com que se mantenham compatíveis os estados destas camadas e dos componentes da Aplicação. A emissão de uma primitiva na camada de Aplicação ou a chegada de uma APDU na camada de Sessão pode afetar o estado destas conexões e associação, alterando os eventos que podem acontecer em seguida. Um exemplo foi mostrado no capítulo 5 para o caso do uso do serviço P-RESYNCHRONIZE.

A falta de sincronismo entre o CCR e seu usuário fez com que algumas alterações tivessem que ser feitas na tabela de estados do protocolo. Nestas alterações foram incluídos alguns estados novos, para o controle do seqüenciamento das primitivas emitidas pelo usuário, algumas células vazias foram preenchidas, nas quais as primitivas emitidas pelo usuário eram descartadas, e algumas transições foram atualizadas, para a incorporação dos novos estados. Se for considerado que o usuário não causa erro de protocolo, estes novos estados podem ser retirados.

Quanto ao uso de ESTELLE, esta linguagem fornece como vantagem sobre uma implementação em C, considerando-se os objetivos didáticos do SISDI-OSI, gerar uma especificação que reflete diretamente a estrutura de comunicação e das mensagens definidas nos protocolos, ou seja, a especificação fica a um nível bastante adequado de abstração, bem próximo do que aparece nos documentos da ISO. Com isto se consegue uma fácil transição entre as especificações nos documentos fornecidos pela ISO e as especificações de implementação. Esta transição facilita o entendimento do protocolo em si e contribui para retirar ambigüidades das especificações informais. O código também tende a ficar mais fácil de ser mantido.

A implementação de um protocolo em ESTELLE possibilita a utilização de ambientes de desenvolvimento de implementações mais poderosos, por basearem em uma linguagem de alto nível de abstração, mais próxima do tipo de solução para o problema. Nestes ambientes podem ser feitas, por exemplo, simulações para se confirmar a correção da especificação de um protocolo (pode-se

modificar parâmetros durante a execução para testar o comportamento do protocolo, acompanhar quais instâncias de módulos estão ativas, quais transições estão habilitadas em um dado instante, etc.).

As desvantagens do uso desta linguagem são a necessidade de se fazer a conversão das mensagens dos canais para a estrutura definida no SISDI-OSI e vice-versa. Pelo fato de os dados em ESTELLE serem declarados em PASCAL, as estruturas usadas de outros componentes tiveram que ser redefinidas, sendo que esta redefinição teve que ser feita tomando-se precauções para que as estruturas não ficassem incompatíveis com as usadas em outros componentes. Para o CCR e o SACF implementado, no entanto, não houve problemas nesta compatibilização. Além disto há a desvantagem de o código em C gerado pelo compilador ser bem maior do que se fosse feito diretamente nesta linguagem (e mais ineficiente). Não puderam ser feitas também as conversões entre primitivas e APDUs, quando possuem parâmetros e campos iguais, como comentado anteriormente, devido à existência do módulo de interface nas especificações usadas na implementação.

Eficiência não é, no entanto, um objetivo principal do sistema. Decisões de implementação que entrem em conflito entre questões de eficiência e estruturação devem ser tomadas considerando o aspecto de estruturação como o mais importante. O código gerado deve possuir níveis razoáveis de eficiência.

A camada de Aplicação é bastante complexa, com vários protocolos, que podem unir-se formando protocolos maiores com relacionamentos complexos. Esta dissertação procurou, portanto, tentar diminuir a complexidade de implementação de protocolos desta camada, formando uma descrição de uma estrutura que seja mais próxima de implementações do que a apresentada em [ISO9545]. Procura, com a definição de funcionalidades, também facilitar a definição de Interfaces de Protocolos de Aplicação, muitas das quais estão sendo definidas recentemente [Haz92].

A definição de funcionalidades para esta camada tenta abranger os requisitos de uma camada de Aplicação bastante complexa, com vários protocolos e contextos de aplicação. Em sistemas mais simples, no entanto, algumas funcionalidades se simplificam. Como exemplo pode-se citar o próprio SISDI-OSI, que só permite algumas concatenações e, assim mesmo, indicadas pelo CCR.

Sugestões para Trabalhos Futuros

Como continuação deste trabalho pode-se sugerir :

1. a integração de funções de gerenciamento neste modelo de funcionalidades;
2. a integração de funcionalidades que não estão presentes na estrutura da camada fornecida pela ISO, como, por exemplo, questões de segurança, modo de comunicação sem conexão e comunicação multi-par;
3. a criação de modelos de implementação, nos quais se descrevam interações mais funcionais entre os componentes, revelando mais precisamente os dados, formas de sincronização entre os componentes, etc.;
4. a integração deste modelo (e de modelos de implementação) com modelos propostos para as Interfaces de Protocolos de Aplicação, como o descrito em [Mad90] e [Mad92].

Apêndice A

Características de ESTELLE usadas nas Implementações

O uso de linguagens naturais para a descrição de sistemas de processamento de informação apresenta problemas, como falta de precisão e presença de ambigüidades. Estes problemas surgem das características próprias deste tipo de linguagens. Por isto foram criadas *Técnicas de Descrição Formal*, que são métodos para se descrever o comportamento de sistemas, de maneira a obter uma descrição completa, consistente, concisa, precisa e sem ambigüidades [ISO9074]. Baseando-se em algum princípio formal, estas técnicas podem ser utilizadas nas fases de projeto, análise, especificação e implementação de sistemas, permitindo que as descrições geradas possam ser analisadas por técnicas também formais. Grande auxílio trazem, portanto, para a fase de análise de correção destas especificações.

ESTELLE é uma das técnicas de descrição formal padronizadas pela ISO [ISO9074]. Utilizada em geral para a descrição formal de sistemas de processamento de informação distribuídos e concorrentes, é utilizada também para se descrever formalmente as definições dos serviços e especificações dos protocolos do RM-OSI. Embora seu objetivo principal seja especificação, o próprio documento da ISO que a descreve, [ISO9074], comenta que as descrições feitas nesta linguagem podem ser usadas como base para implementações, utilizando-se ferramentas de geração automática de código. Foi para este fim que ESTELLE foi utilizada nesta tese.

O modelo no qual ESTELLE se baseia é o de uma máquina de estados finita estendida com recursos como o uso de variáveis e prioridade nas transições. A linguagem permite que se especifiquem máquinas de estado cujas ações internas sejam descritas por instruções em PASCAL (com restrições e extensões a esta linguagem).

Nesta seção serão apresentados os aspectos de ESTELLE necessários à compreensão do capítulo 5, que apresenta as implementações desta tese. Este anexo se baseia em [ISO9074] e [Bud87], que descreve as principais características da linguagem. [Linn87] também aborda os principais aspectos da linguagem e apresenta uma série de exemplos de especificações. Para maiores esclarecimentos, indicam-se estas referências.

A.1 O Modelo Geral de ESTELLE

Uma especificação em ESTELLE define um conjunto de módulos estruturados hierarquicamente. Durante a execução da especificação, várias instâncias destes módulos podem ser criadas, o que gera uma hierarquia de instâncias dos módulos. A figura A.1 ilustra tais hierarquias. Estas instâncias podem-se comunicar através de *canais* bidirecionais, associados a *pontos de interação*. Estes pontos de interação representam por onde as instâncias de módulos enviam e recebem mensagens, chamadas em ESTELLE de *interações*. Uma instância de módulo pode possuir variáveis que, além de por ela própria, podem também ser lidas e alteradas por sua instância de módulo pai, ou seja, a instância de módulo imediatamente acima na estrutura hierárquica. Um módulo deve ser considerado como uma *caixa-preta* e compõe-se de uma definição de cabeçalho, que define a interface externa desta caixa-preta, e de um corpo, que descreve como será o comportamento de suas instâncias. Este comportamento é definido através de dois conjuntos de transições: um que só será analisado assim que uma instância for criada, e serve para descrever ações a serem realizadas neste instante, e um outro, que define o comportamento de uma instância deste módulo durante o seu tempo de vida. Ambos os conjuntos são opcionais. Se um módulo possuir o segundo, o módulo será dito *ativo*; senão, será dito *inativo*. Em ESTELLE mais de uma definição de corpo pode estar associada a uma mesma definição de cabeçalho. Cada par formado por uma definição de corpo e uma definição de cabeçalho forma um módulo.

A uma definição de cabeçalho pode-se atribuir, mas não necessariamente, um dos seguintes atributos : *SYSTEMPROCESS*, *SYSTEMACTIVITY*, *PROCESS* e *ACTIVITY*. Estes atributos servem para definir o comportamento das instâncias dos módulos em termos de paralelismo e sincronismo. As regras para esta atribuição são:

1. todo módulo ativo deve possuir um atributo;
2. um módulo com atributo SYSTEMPROCESS ou SYSTEMACTIVITY não pode estar em um nível na hierarquia abaixo de um outro módulo com atributo (qualquer atributo que seja);
3. módulos com atributo PROCESS ou ACTIVITY têm que possuir um módulo com atributo SYSTEMPROCESS ou SYSTEMACTIVITY em algum nível superior na hierarquia;
4. um módulo com atributo PROCESS ou SYSTEMPROCESS só pode ser sub-estruturado em módulos com atributo PROCESS ou ACTIVITY. Juntamente com o item 2 acima, isto significa que módulos sem atributos são somente aqueles módulos inativos que englobam módulos com atributo SYSTEMPROCESS ou SYSTEMACTIVITY, se tais módulos existirem;
5. um módulo com atributo ACTIVITY ou SYSTEMACTIVITY só pode ser sub-estruturado em módulos com atributo ACTIVITY.

O prefixo SYSTEM nos atributos SYSTEMPROCESS e SYSTEMACTIVITY serve para definir sub-sistemas hierárquicos de instâncias de módulos que executam em paralelo, de maneira completamente independente umas das outras. Os atributos PROCESS e ACTIVITY (ou os sufixos PROCESS e ACTIVITY nos atributos SYSTEMPROCESS e SYSTEMACTIVITY) servem para descrever duas formas de execução das instâncias de um sistema: paralelismo sincronizado (PROCESS) ou não determinismo (ACTIVITY). Eles influenciam a determinação do conjunto de transições das instâncias filhas de uma instância de módulo a serem executadas. Associada ao fato

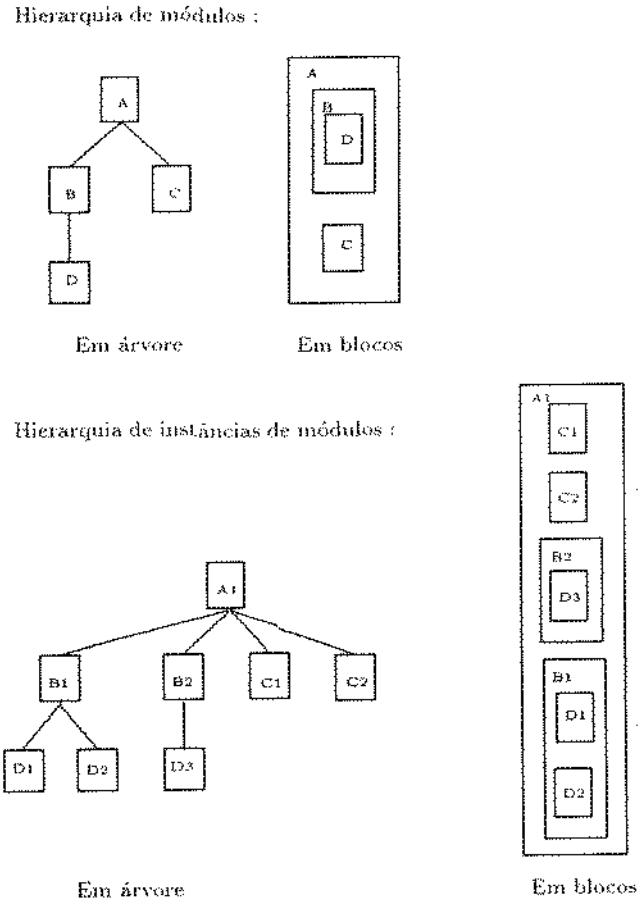

Figura A.1: Hierarquia de módulos e instâncias de módulos ESTELLE

de uma transição em uma instância de módulo possuir prioridade sobre transições de todas as suas instâncias descendentes, a influência dos atributos é melhor explicada com o exemplo que aparece em [Bud87], aqui reproduzido na figura A.2.

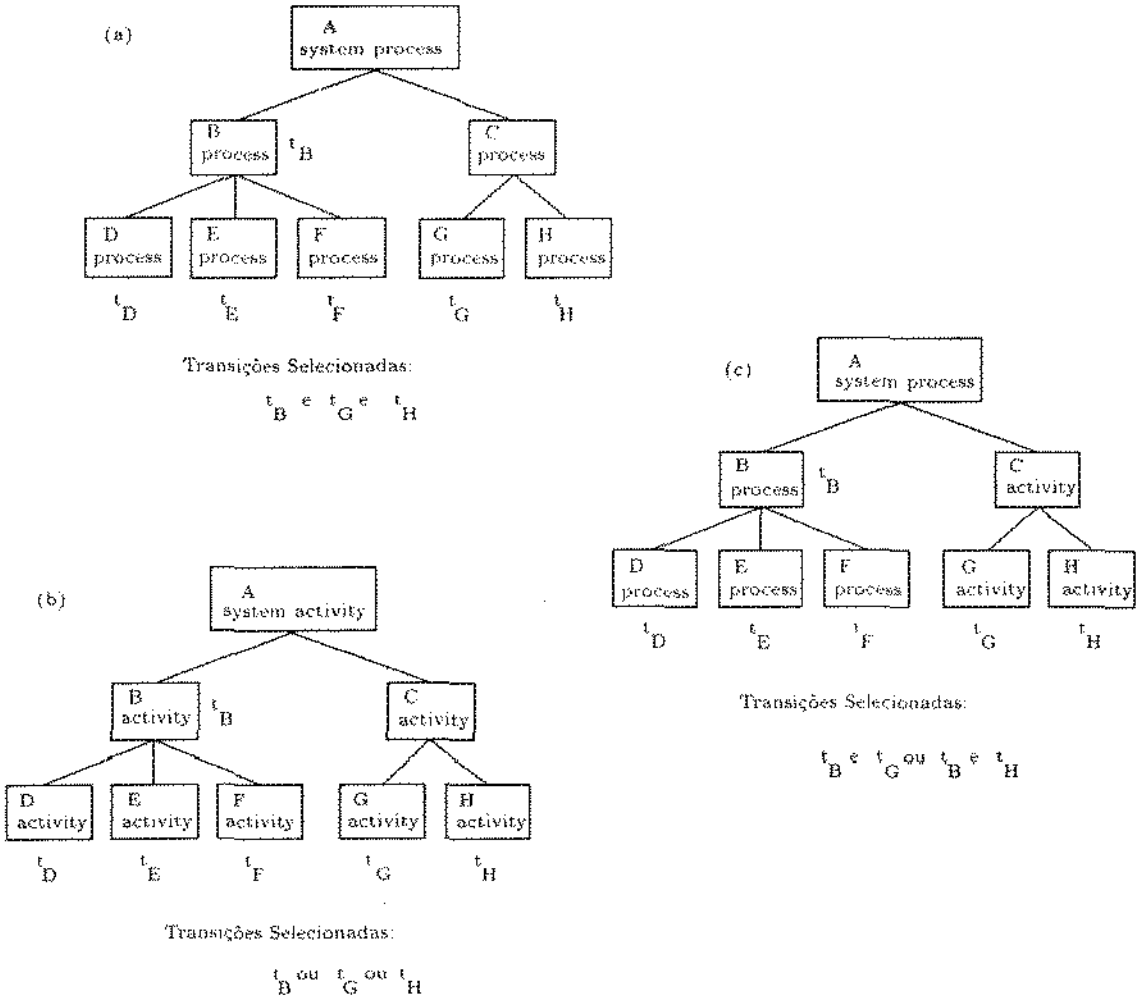

Figura A.2: Efeito das classes dos módulos na seleção de transições

Um atributo PROCESS (ou SYSTEMPROCESS) faz com que todas as transições oferecidas por suas instâncias filhas) sejam executadas, o que provê um paralelismo sincronizado entre estas instâncias. Só pode haver uma transição por instância a cada vez. No caso de ACTIVITY (ou SYSTEMACTIVITY), somente uma das transições oferecidas por todas as suas instâncias filhas será escolhida para ser executada. Esta escolha é feita de modo arbitrário, o que provê uma forma de execução não determinística.

Uma vez determinado este conjunto de transições para cada um dos subsistemas, todas elas serão executadas em paralelo. Quando todas as transições para um determinado subsistema já tiverem executado, e somente neste instante, um novo conjunto será determinado.

Cada instância de módulo contém um conjunto de transições. Cada transição possui um con-

junto de cláusulas. A transição oferecida por uma instância de um módulo é uma das transições que possuem todas as suas cláusulas satisfeitas. A escolha da transição específica a se oferecer é feita de modo aleatório.

A.2 Alguns Aspectos de Sintaxe da Linguagem

Nesta seção serão apresentadas as construções sintáticas de ESTELLE usadas na implementação do protocolo CCR e do SACF. Esta exposição apresenta apenas um sub-conjunto das capacidades da linguagem.

A.2.1 Definição de Canais e Declaração de Pontos de Interação

A figura A.3 ilustra a maneira como os canais de comunicação são definidos em ESTELLE. Nesta figura *id_canal* é o identificador do canal. *m1..mL* e *n1..nK* são definições de tipos de interações (mensagens) e seus parâmetros que podem ser enviadas através do canal. A especificação de uma interação é análoga à declaração de subrotinas em linguagens como PASCAL. Definem-se o número de parâmetros e seus tipos. A cada vez que uma interação de um determinado tipo for enviada em um canal, valores reais para estes parâmetros serão fornecidos.

```
CHANNEL id_canal (funcao1,funcao2);

 BY funcao1 : m1;
              m2;
               .
               .
               .
              mL;

 BY funcao2 : n1;
              n2;
               .
               .
               .
              nK;
```

Figura A.3: Declaração de canais

Os identificadores *funcao1* e *funcao2* servem para se especificar quais interações podem ser enviadas em cada sentido do canal. A declaração de dois pontos de interação, por exemplo, *p1* e *p2*, que se comunicarão através do canal definido na figura A.3, é feita da seguinte maneira:

p1: id_canal(funcao1)
e
p2: id_canal(funcao2)

Estas declarações indicam que *p1* e *p2* podem trocar interações com outros pontos de interação através de um canal do tipo *id_canal*. Indicam também que p1 só pode enviar as interações associadas ao identificador *funcao1* na definição do canal *id_canal*, ou seja, *m1..mL*, e só pode receber as interações associadas ao papel oposto na definição deste canal, ou seja, *funcao2*. As interações associadas a este tipo são *n1..nK*. Com *p2* ocorre o inverso: só pode enviar as interações associadas a *funcao2* na definição do canal *id_canal*, ou seja, *n1..nK*, e só receber as associadas ao papel inverso, ou seja, as interações *m1..mL*.

Canais unidirecionais podem ser especificados declarando-se duas funções, mas atribuindo interações a apenas um deles. Para se especificar interações que podem ser enviadas nos dois sentidos, coloca-se, após a palavra-chave *BY*, os dois identificadores de função, separados por vírgula.

A cada ponto de interação está associada uma fila de recepção de mensagens. Nas declarações de cada um deles pode-se ainda especificar se esta fila é exclusiva do ponto ou se ela pode ser compartilhada com outros pontos. No primeiro caso especifica-se o ponto com as palavras-chave *individual queue*; no segundo, com *common queue*, como, por exemplo, nas declarações abaixo:

p1: id_canal (funcao1) common queue
e
p2: id_canal (funcao2) individual queue

Como exatamente o compartilhamento da fila é feito não é de interesse para este trabalho e não será comentado portanto.

A.2.2 Cabeçalho e Corpo de Módulos

Cada módulo é constituído de uma definição de um cabeçalho e de um corpo. A estrutura para a definição de um cabeçalho é de acordo com a que aparece na figura A.4.

```
module A systemprocess (n:integer);
  ip p: T(S) individual queue;
     p1: U(S) common queue;
     p2: W(K) common queue;
  export X,Y: integer; Z: boolean
end;
```

Figura A.4: Cabeçalho de módulo

A primeira parte da definição consiste da palavra-chave *module*, seguida de um identificador para o cabeçalho, a classe do módulo e de uma lista de parâmetros. O identificador do cabeçalho (*A*, no exemplo) serve para se referir ao cabeçalho em outras partes da especificação. A lista de parâmetros é análoga a de uma subrotina em PASCAL, mas só se pode passar parâmetros por valor. Estes parâmetros servem para que informações sejam passadas a uma instância de módulo, no momento de sua criação. Os tipos de classe que se pode atribuir ao cabeçalho são os descritos anteriormente, ou seja, SYSTEMPROCESS, SYSTEMACTIVITY, PROCESS e ACTIVITY.

Após a definição dos parâmetros há uma parte para a declaração de pontos de interação e uma parte para a declaração das variáveis a serem exportadas, ou seja, aquelas que podem ser acessadas

pelo módulo pai. A primeira destas partes é formada pela palavra-chave *ip*, seguida da lista de declarações dos pontos de interação. Esta declaração é feita da maneira comentada anteriormente. Estes pontos de interação, declarados no cabeçalho de um módulo, são chamados de *pontos de interação externos*. Estes pontos são usados, na estrutura de implementação do CCR e do SACF, para a comunicação entre instâncias de módulos filhos de uma mesma instância de módulo pai.

A segunda parte consiste da palavra chave *export*, seguida de uma lista de declaração de variáveis, feita exatamente da mesma maneira que em um programa PASCAL. Estas variáveis podem ser lidas e alteradas pela instância de módulo pai do módulo que as declara. Os tipos destas variáveis têm que estar definidos em um módulo ancestral, e serem obedecidas regras de escopo análogas às de procedimentos e funções de PASCAL.

Enquanto o cabeçalho de um módulo define a sua interface externa, ou seja, os pontos de interação externos e as variáveis exportadas, a definição do corpo de um módulo vai especificar qual será o seu comportamento, ou seja, é usada para especificar a máquina de estados do módulo. Este comportamento é especificado por um conjunto de transições. Mais de um corpo pode estar associado a um mesmo cabeçalho, formando, com isto, módulos com mesma interface externa, mas de comportamentos diferentes.

A figura A.5 mostra a estrutura geral de definição do corpo de um módulo.

```
body B for A;
   "definicao do corpo"
end;
```

Figura A.5: Corpo de módulo

Nesta definição há a palavra-chave *body*, seguida de um identificador para o corpo, que é *B* no exemplo, pela palavra-chave *for* e pelo identificador do cabeçalho ao qual esta definição de corpo está relacionada, no caso, ao cabeçalho *A*, definido na figura A.4. A seguir vem a definição em si do corpo do módulo, que é composto das três partes seguintes: uma parte de declarações, uma parte de inicialização e uma parte de transições.

A.2.3 Declarações

A parte de declarações é usada para se declarar objetos próprios de ESTELLE e fazer declarações de um programa comum de PASCAL, que são constantes, tipos, variáveis, funções e procedimentos. As funções e procedimentos possuem estruturas análogas às de PASCAL e podem ser escritas na própria especificação ESTELLE ou ter apenas um cabeçalho e, ao invés de um corpo, conter uma diretiva. Uma destas diretivas é a *PRIMITIVE*, que indica que a definição do corpo da função ou procedimento não é dada no texto da especificação. Esta diretiva é útil, no nível de especificação de um sistema, para apenas indicar o que uma função ou procedimento faz, sem escrevê-la realmente. No nível de implementação é útil para permitir que as rotinas sejam feitas em outras linguagens e apenas *link-editadas* com o código gerado a partir da especificação em ESTELLE. Os objetos de ESTELLE declarados nesta parte são canais, novos módulos, variáveis de módulo, estados, conjuntos de estados e pontos de interação internos.

As declarações de canais e pontos de interação já foram descritas anteriormente. Os módulos declarados dentro de um outro módulo constituem filhos deste último, formando com isto a hie-

rarquia de módulos. Os pontos de interação declarados nesta parte de definição de corpo de um módulo vão constituir pontos de interação *INTERNOS*. Estes pontos são usados para se trocar mensagens internamente a uma instância de módulo ou para trocar mensagens entre uma instância de módulo e suas instâncias filhas. Estes pontos não são usados na implementação do CCR.

Variáveis de módulo são variáveis usadas para referenciar instâncias de módulos. As declarações são feitas de modo análogo à que aparece na figura A.6, onde há a definição de três variáveis de módulo. Neste exemplo as variáveis podem referenciar instâncias de módulos cujo cabeçalho tenha o identificador *A*. Este cabeçalho é o definido na figura A.4. Observe que as variáveis de módulo possuem como tipo o identificador do cabeçalho e não do corpo do módulo.

```
modvar X, Y, Z: A;
```

Figura A.6: Declaração de variáveis de módulo

A declaração de estados serve para indicar os estados que irão compor a máquina de estados que a especificação ESTELLE representa. Esta declaração consiste apenas da palavra-chave *state* seguida dos nomes dos estados, ou da palavra-chave *stateset* seguida de declarações de conjuntos de estados. A figura A.7 ilustra estas declarações.

```
state IDLE, WAIT, OPEN, CLOSED;
stateset IDWA = (IDLE,WAIT);
```

Figura A.7: Estados e conjuntos de estados

Nesta figura são declarados quatro estados, *IDLE*, *WAIT*, *OPEN* e *CLOSED* e um conjunto de estados, *IDWA*, que consiste dos estados *IDLE* e *WAIT*. A declaração de conjuntos de estados serve para se referir a vários estados de uma única vez. É usado para expressar que uma determinada transição está habilitada se a máquina de estado estiver em qualquer um dos estados especificados no conjunto. Isto evita ter que especificar a transição para cada um deles separadamente.

A.2.4 Transições

A parte de inicialização e a parte de transições consistem de uma declaração de um conjunto de transições. A diferença que há entre estas partes é que a primeira define as transições que podem ocorrer no momento de criação de uma instância do módulo, enquanto que a segunda descreve as transições que podem ocorrer após a inicialização, durante a execução normal da especificação. Há algumas restrições impostas sobre as estruturas das transições da parte de inicialização, mas que não serão comentadas aqui.

A parte de transições consiste de um conjunto de transições que vão especificar o comportamento do módulo, indicando em que situações as mudanças de estado podem ocorrer e quais ações devem ser realizadas nestas transições. Como o objetivo desta seção é mostrar as partes de ESTELLE necessárias ao entendimento das implementações desta tese, será mostrada apenas a parte da estrutura de declaração de transições que foi utilizada.

A estrutura geral das transições utilizada é mostrada na figura A.8, e consiste de uma declaração de cláusulas e de uma parte de especificação de instruções. As cláusulas especificam em que situação a transição fica habilitada e qual será o estado para o qual a máquina irá, enquanto as instruções indicam o que será executado quando uma destas transições ocorrer. Somente se todas as cláusulas forem satisfeitas é que a transição estará habilitada.

```
trans <clausulas>
      <instrucoes>
```

Figura A.8: Transições

As cláusulas que podem estar presentes são:

1. *FROM* <estados>

   indica os estados nos quais a transição pode ocorrer (e as ações então serem executadas);

2. *TO* <estado>

   indica para qual estado a máquina de estado irá após a execução da transição. Se o estado for *SAME*, o estado da máquina não muda. Nas transições da parte de inicialização, é usada para especificar o estado inicial da máquina;

3. *WHEN* <ponto de interação>.<interação>

   indica que a transição só poderá estar habilitada se houver uma interação do tipo especificado em <interação> como primeiro elemento da fila correspondente ao ponto de interação determinado em <ponto de interação>. Esta cláusula corresponde ao teste de chegada de uma interação na fila associada ao ponto;

4. *PROVIDED* <expressão booleana>

   possui uma expressão booleana que expressa uma condição que deve ser satisfeita para a transição poder ocorrer.

   ESTELLE fornece a expressão *EXIST*, que pode ser usada na parte da expressão booleana desta cláusula. Na implementação do CCR e do SACF, esta expressão foi usada com a estrutura mostrada na figura A.9. Nesta figura X é uma variável de módulo e *id_cabecalho* é um identificador de cabeçalho. A expressão EXIST é uma expressão booleana que, com esta estrutura apresentada, possui o valor TRUE se a expressão booleana <expressao booleana> for verdadeira para alguma das instâncias filhas cuja definição de cabeçalho seja identificada pelo identificador *id_cabecalho*. Se houver alguma, a variável X referenciará uma qualquer destas instâncias e poderá ser usada, por exemplo, para acessar variáveis exportadas por esta instância;

5. *ANY* <lista de domínios> *DO*

   esta é uma cláusula que fornece um modo abreviado de se escrever uma série de transições. A *<lista de domínios>* representa uma lista de declarações em que cada componente é uma seqüência de identificadores e um tipo ordinal. A expansão de uma cláusula ANY pode ser

```
EXIST X : id_cabecalho
 SUCHTHAT <expressao booleana>
```

Figura A.9: Cláusula EXIST

```
trans
  when ip.m
   from A
    any X : 1..2 do
     provided E(X)
      to C
       begin
        S11(X); S12(X)
       end;

Expansao:

trans
  when ip.m
   from A
    provided E(1)
     to C
      begin
       S11(1); S12(1)
      end;

trans
  when ip.m
   from A
    provided E(2)
     to C
      begin
       S11(2); S12(2)
      end;
```

Figura A.10: Expansão de uma cláusula ANY

vista no exemplo da figura A.10. Cada ocorrência dos identificadores à esquerda dos dois pontos (:) na cláusula ANY são substituídos pelos valores possíveis para o tipo à direita.

Em ESTELLE há mais dois tipos de cláusula, a cláusula *DELAY* e a cláusula *PRIORITY*, mas que não foram usadas na implementação do CCR. A figura A.10 serve também para exemplificar transições.

As transições podem estar *aninhadas* também. O aninhamento de transições constitui apenas uma outra maneira de escrever mais de uma transição de forma abreviada, na qual algumas cláusulas de uma transição são herdadas por transições escritas depois dela. O exemplo da figura A.11 ilustra como as duas transições expandidas da figura A.10 poderiam ser escritas de modo aninhado. Observe, por este exemplo, que as transições herdam das transições anteriores as cláusulas escritas antes da primeira cláusula que têm em comum. A palavra-chave *trans* termina este processo de herança.

```
trans
  when ip.m
   from A
    provided E(1)
     to C
      begin
       S11(1); S12(1)
      end;

    provided E(2)
     to C
      begin
       S11(2); S12(2)
      end;
```

Figura A.11: Transições aninhadas

A parte de instruções pode conter algumas instruções da linguagem PASCAL e instruções próprias de ESTELLE. ESTELLE só permite um sub-conjunto das instruções de PASCAL. Este sub-conjunto está especificado em [ISO9074]. As instruções de ESTELLE de interesse para este texto são mostradas a seguir :

init serve para criar uma instância de um módulo. Por exemplo

init X with B(3)

cria uma nova instância do módulo com corpo *B*. *X* é uma variável de módulo que será usada para se referenciar à instância criada. Uma referência a uma instância pode ser feita, por exemplo, por uma instância de módulo para acessar as variáveis externas de uma instância de módulo filha. Após o identificador do corpo, no exemplo, *B*, há a passagem de parâmetros. Este exemplo segue as definições de cabeçalho e corpo de módulo mostradas nas figuras A.4 e

A.5. Nestas definições só foi especificado um parâmetro, de valor inteiro. No exemplo acima atribuiu-se o valor 3 a este parâmetro. Este valor poderá ser usado pela instância criada.

Observe que após a palavra-chave *with* está o nome de um corpo de módulo, enquanto a variável *X* é declarada como do tipo *A* (figura A.4), que é um identificador de um cabeçalho. A lista de parâmetros fica especificada também na declaração do cabeçalho.

A referência a uma variável exportada por uma instância de um módulo é feita usando-se o nome da variável de módulo, seguido de um ponto e o nome da variável externa, como em :

X.nome_da_variavel,

onde *X* é uma variável de módulo e *nome_da_variavel* é o nome da variável exportada da instância. Referências aos pontos de interação externos são feitas de maneira análoga;

connect serve para conectar dois pontos de interação, mas que sejam dos seguintes tipos:

- ou dois pontos de interação externos de instâncias de módulos filhos;
- ou dois pontos de interação internos;
- ou um ponto de interação interno com um ponto de interação externo de uma instância filha.

Um exemplo pode ser:

connect p1 to p2

onde *p1* e *p2* devem ser dois pontos de interação de um dos tipos acima. Além disto *p1* e *p2* têm que ter sido declarados com o mesmo nome de canal, mas de funções opostas, como feito no exemplo da seção A.2.1.

As interações associadas à função *funcao1* podem então ser enviadas através de *p1*, que serão recebidas em *p2*. De modo análogo, as interações associadas ao papel *papel2* podem ser enviadas por *p2* e serão recebidas em *p1*. Existe uma fila em cada ponto de interação para a recepção das interações.

output esta é a instrução usada para se enviar interações entre as instâncias de módulos, através dos pontos de interação. Esta instrução é utilizada como no exemplo seguinte:

output p1.interacao (par1, par2, par3)

onde *p1* especifica um ponto de interação, *interacao* é o nome da interação a ser enviada e *par1*, *par2* e *par3* são valores para os parâmetros da interação. Para que esta instrução possa ser usada, *p1* tem que ter sido declarado com um nome de um canal e de uma função que contenha a descrição desta interação.

release e terminate estas duas instruções são usadas como nos exemplos abaixo :

release X
terminate X

onde X é uma variável de módulo. Estas instruções são usadas por uma instância de módulo para terminar uma instância de módulo filha, referenciada por X. Todos os pontos de interação externos desta instância são desconectados e todas as suas instâncias filhas também são terminadas da mesma maneira. O procedimento se expande recursivamente. A diferença entre as duas instruções não influencia no uso destas instruções na implementação do CCR. Qualquer uma das duas pode ser usada.

all esta é uma instrução usada, nas implementações desta dissertação, para se iteragir sobre um conjunto de instâncias de módulos. Um exemplo da estrutura usada é

```
ALL instancia : id_cabecalho_modulo DO
  <instrucoes>
```

Neste exemplo se fará uma iteração sobre todas as instâncias de módulos cujas definições de cabeçalho possuam o identificador *id_cabecalho_modulo*, de modo análogo ao caso da expressão *EXIST*, comentada anteriormente. O conjunto de instruções especificadas em <*instrucoes*> será executado para cada uma das instâncias, que são referenciadas utilizando a variável *instancia*.

A.2.5 Inicialização

A parte de inicialização é identificada pela palavra-chave *INITIALIZE* e consiste também de um conjunto de transições, como comentado anteriormente. No momento de criação de uma instância de módulo, uma das transições deste conjunto que estiver habilitada será executada. A escolha é feita de modo arbitrário.

Esta parte pode ser usada para, por exemplo, definir valores iniciais para variáveis, ou criar uma configuração inicial de instâncias de módulos filhas e conectar seus pontos de interação. Se foram declarados estados para o módulo, as transições têm que, necessariamente, ter uma cláusula *TO*, para especificar o estado inicial da máquina de estados que a especificação ESTELLE representa.

A.2.6 O Módulo de Especificação

Como dito anteriormente, uma especificação em ESTELLE consiste de uma hierarquia de módulos. O módulo mais geral, que engloba todos os outros, é especificado de um modo especial. A sua estrutura é mostrada na figura A.12.

Nesta figura <*nome_especificação*> é o nome usado para identificar toda a especificação. <*system_class*> é usado para especificar um dos atributos SYSTEMPROCESS ou SYSTEMACTIVITY, da mesma maneira que comentado anteriormente. <*opcoes*> é utilizado para especificar um padrão *default* para a política das filas, se *common queue* ou *individual queue*, e para especificar, quando for necessário, uma unidade para medidas de tempo. Estas opções não são usadas na implementação do CCR. *definição de corpo* é exatamente do mesmo modo que a parte de mesmo nome comentada anteriormente. As declarações dos módulos do primeiro nível da hierarquia são feitas nesta parte. Os outros módulos são declarados internamente a estes.

```
specification <nome_especificacao> <system_class>;
  <opcoes>
  "definicao de corpo"
end.
```

Figura A.12: Estrutura de uma especificação em ESTELLE

Bibliografia

[Amb] Lisiane Maria Bannwart Ambiel. Tese de Mestrado em andamento.

[Bac86] Maurice J. Bach. *The Design of the UNIX Operating System*. Prentice-Hall Software Series. Prentice-Hall, Inc., EUA, 1986.

[Ber87] Philip A. Bernstein, Vassos Hadzilacos e Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publishing Company, EUA, 1987.

[Bla89] Uyless Black. *Data Networks Concepts. Theory, and Practice*. Prentice-Hall International, Inc., EUA, 1989.

[Boc87] Gregor v. Bochmann, George Walter Gerber e Jean-Marc Serre. Semiautomatic Implementation of Communication Protocols. *IEEE Transactions on Software Engineering*, SE-13(9):989–1000, setembro 1987.

[Bud87] S. Budkowski e P. Dembinski. An introduction to Estelle: A Specification Language for Distributed Systems. *Computer Networks and ISDN Systems*, 14:3–23, 1987.

[CCITT216] CCITT. Presentation Service Definition for Open Systems Interconnection for CCITT Applications – Recommendation X.216, 1988.

[CCITT217] CCITT. Association Control Service Definition for Open Systems Interconnection for CCITT Applications – Recommendation X.217, 1988.

[CCITT226] CCITT. Presentation Protocol Specification for Open Systems Interconnection for CCITT Applications – Recommendation X.226, 1988.

[CCITT227] CCITT. Association Control Protocol Specification for Open Systems Interconnection for CCITT Applications – Recommendation X.227. 1988.

[CCITT500] CCITT. Data Communication Networks Directory – Recommendations X.500-X.521, 1989.

[Chi90] Sanjay B. Chikarmane. Upper Layer Architecture for HP MAP 3.0 OSI Services. *Hewlett-Packard Journal*, 11–14, agosto 1990.

[Coo90] Beth E. Cooke. Colleen S. Fettig, Paul B. Koski, Darrell O. Swope e Roy M. Vandoorn. Directory Services in the HP MAP 3.0 Environment. *Hewlett-Packard Journal*, 15–23, agosto 1990.

[EWS89] *EWS User's Manual - Esprit Project 265 - SEDOS Estelle Demonstrator*, junho 1989.

[Fuj] Célio Toshihiro Fujito. Tese de Mestrado em andamento.

[Hal88] Fred Halsall. *Data Communications, Computer Networks and OSI*. Addison-Wesley Publishing Company, EUA, segunda edição, 1988.

[Haz92] Mark Hazzard. Distributed Services within IEEE TCOS/POSIX. *IEEE Network*, 22–23, março 1992.

[Hen90] John Henshall e Sandy Shaw. *OSI Explained: End-to-end Computer Communication Standards*. Ellis Horwood Series in Computer Communications and Networking, Ellis Horwood, Inglaterra, segunda edição, 1990.

[Ina90] Paulo César Minoru Inazumi. *Aspectos de Especificação e Implementação da Camada de Apresentação do Padrão MAP utilizando o ambiente EPOS*. Tese de Mestrado, DCA - FEE - UNICAMP, Campinas, SP, novembro 1990.

[ISO7498] ISO. Information Processing Systems – Open Systems Interconnection – Basic Reference Model – ISO 7498, 1984.

[ISO7498-3] ISO. Information Processing Systems – Open Systems Interconnection – Basic Reference Model – Part 3: Naming and Addressing ISO 7498-3, 1989.

[ISO8326] ISO. Information Processing Systems – Open Systems Interconnection – Basic Connection oriented Session Service Definition – ISO 8326, agosto 1987.

[ISO8327] ISO. Information Processing Systems – Open Systems Interconnection – Basic Connection oriented Session Protocol Specification – ISO 8327, agosto 1987.

[ISO8571] ISO. Information Processing Systems – OSI – File Transfer, Access and Management – Part I : General Description, Part II : The Virtual Filestore, Part III : The File Service Definition, Part IV : The File Protocol Specification – ISO DP-8571 – FTAM 2nd DP, 1985.

[ISO8824] ISO. Information Processing Systems – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1) – ISO 8824, dezembro 1987.

[ISO8825] ISO. Information Processing Systems – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) – ISO 8825, novembro 1987.

[ISO9074] ISO. Information Processing Systems – Open Systems Interconnection – Estelle - a Formal Description Technique based on an Extended State Transition Model – ISO/IEC ISO 9074, novembro 1988.

[ISO9545] ISO. Information Processing Systems – Open Systems Interconnection – Application Layer Structure – ISO/IEC DIS 9545, 1988.

[ISO9579-1] ISO. Information Processing Systems – Open Systems Interconnection – Remote Database Access – Part 1 : Generic Model, Service and Protocol – ISO 9579-1, maio 1991.

[ISO9804] ISO. CCR Service Definition – ISO 9804 – final text, fevereiro 1990.

[ISO9805] ISO. Information Technology – Open Systems Interconnection – Protocol Specification for the Commitment, Concurrency and Recovery Service Element – ISO/IEC 9805, abril 1990.

[ISO10026-1] ISO. Information Technology – Open Systems Interconnection – Distributed Transaction Processing – Part 1: OSI TP Model – ISO/IEC 10026–1, maio 1991.

[ISO10026-2] ISO. Information Technology – Open Systems Interconnection – Distributed Transaction Processing – Part 2: OSI TP Service – ISO/IEC DIS 10026–2, maio 1991.

[ISO10026-3] ISO. Information Technology – Open Systems Interconnection – Distributed Transaction Processing – Part 3: Protocol Specification – ISO/IEC 10026–3, outubro 1991.

[Jac89] Durval Carvalho de Ávila Jacintho. *Aspectos de Especificação e Implementação da Estrutura de Mensagens de um Sistema Didático do Protocolo MMS.* Tese de Mestrado, DCA - FEE - UNICAMP, Campinas, SP, novembro 1989.

[Joh] Wolfgang Johannsen e Winfried Lamersdorf. *An Open Systems Architecture for Transaction Supported Distributed Database Applications.* IBM Deutschland, European Networking Center, Alemanha.

[Knu73] Donald E. Knuth. *The Art of Computer Programming - Volume 1: Fundamental Algorithms.* Addison-Wesley Publishing Company, EUA, segunda edição, 435:451, 1973.

[Lin92] Chen Wen Lin. *Uma Metodologia para Implementação Semi-automática de Protocolos de Comunicação.* Tese de Mestrado, Universidade Estadual da Paraíba, Campina Grande, PB, agosto 1992.

[Linn87] Jr. Richard J. Linn. *A Revised Draft Tutorial on the Features and Facilities of Estelle.* National Bureau of Standards – Institute for Computer Science and Technology – Systems and Network Architecture Division, EUA, agosto 1987.

[Mad90] Edmundo Roberto Mauro Madeira e Manuel de Jesus Mendes. An Application Interface Model for Communication Software. em *IEEE Global Telecommunications Conference – GLOBECOM'90.* San Diego, EUA, dezembro 1990.

[Mad91] Edmundo Roberto Mauro Madeira. *Um Modelo de Interface de Aplicação para Sistemas de Comunicação.* Tese de Doutorado, UNICAMP, 1991.

[Mad92] Edmundo Roberto Mauro Madeira e Manuel de Jesus Mendes. An Application Interface Model for Industrial Networks and its External View. Em *INFONOR'92*, Antofagasta, Chile, dezembro 1992.

[Mdz90] Carlos Raul Arias Mendez. *Aspectos de Implementação do Protocolo de Sessão utilizando Ferramentas CASE.* Tese de Mestrado, DCA - FEE - UNICAMP, Campinas, SP, dezembro 1990.

[Mel86] Wilfred A. Melendez e Erik Lorenz Petersen. The Upper Layers of the ISO/OSI Reference Model (Part I). *Computer Standards and Interfaces*, 5:13–46, 1986.

[Men90] Manuel de Jesus Mendes. *Redes Locais Industriais.* UNICAMP/CONSAI, 1990.

[Men93] Manuel de Jesus Mendes, Edmundo Roberto Mauro Madeira, Flávio Morais de Assis Silva, Elza Kiyomi Shimabukuro Garcia, Lisiane Maria Bannwart Ambiel, Luiz Otávio Merlin Miranda, María Inés Valderrama Restovíc, Milton T. Sakamoto e Célio Toshihiro Fujito. SISDI-OSI: Sistema Didático para o Modelo OSI. Em *Anais do 11o. Simpósio Brasileiro de Redes de Computadores*, Campinas, maio 1993.

[Moh83] C. Mohan e B. Lindsay. Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions. IBM Research Laboratory, San Jose, EUA, 1983.

[Mun91] Cláudio Rodrigues Muniz da Silva. *Implementação do Elemento de Serviço para Operações Remotas utilizando uma Ferramenta de CASE conhecida por EPOS.* Tese de Mestrado, DCA - FEE - UNICAMP, Campinas, SP, junho 1991.

[Pag89] Antenor Paglione Junior, Durval Carvalho Ávila Jacintho, Edmundo Roberto Mauro Madeira, Ivo Alexandre Fernandes, Jaime Nicolato Correa, José Mario Souza Lima, Maria Cristina Zabeu, Verônica Lima Pimentel de Sousa, e Manuel de Jesus Mendes. SISDI-MAP: Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP. Em *Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos*, Florianópolis, SC, setembro 1989.

[Pag91] Antenor Paglione Júnior. *Aspectos de Especificação e Implementação da Estrutura de Mensagens da Máquina de Protocolo ACSE para o SISDI-MAP.* Tese de Mestrado, DCA - FEE - UNICAMP, Campinas, SP, julho 1991.

[Par90] Collin Y. W. Park e Bruce J. Talley. HP Manufacturing Automation Protocol 3.0. *Hewlett-Packard Journal*, pags 6–10, agosto 1990.

[Res92] María Inés Valderrama Restovíc. *Compilador ASN.1 e Codificador/Decodificador BER.* Tese de Mestrado. UNICAMP, setembro 1992.

[Saka] Milton Tsuyoshi Sakamoto. Tese de Mestrado em andamento.

[Sil93] Flávio Morais de Assis Silva e Edmundo Roberto Mauro Madeira. Um Modelo Conceitual para os Componentes da Camada de Aplicação do RM-OSI/ISO e Aspectos de sua Implementação. Em *Anais do 11o. Simpósio Brasileiro de Redes de Computadores.* Campinas, maio 1993.

[SUN90] SUN Microsystems, EUA. *Programming Utilities and Libraries.* Revision A of 27 March 1990.

[Svo89] Liba Svobodova. Implementing OSI Systems. *IEEE Journal on Selected Areas in Communications*, 7(7):1115–1130, setembro 1989.

[Tan88] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall International, Inc., EUA, segunda edição, 1988.

[Vuo88] Son T. Vuong, Allen C. Lau e R. Isaac Chan. Semiautomatic Implementation of Protocols using an Estelle-C Compiler. *IEEE Transactions on Software Engineering*, 14(3):384–393, março 1988.