

Capítulo 7

A CAMADA DE APRESENTAÇÃO

A camada de apresentação é empregada para alterar a representação de dados transmitidos via rede para fins de:

1. compatibilizar a comunicação;
2. segurança e privacidade;
3. compactação de dados.

O primeiro item diz respeito a representação canônica de dados: uma representação inteligível por todos os hosts comunicantes independentemente de suas arquiteturas de máquina. Os dois últimos itens se relacionam com criptografia e compressão de dados. A camada de apresentação é o local para se implementar tais funções. Criptografia e compressão de dados são tópicos extensos e dependentes do contexto no qual a aplicação está inserida. Estes tópicos não serão cobertos neste texto.

7.1 Representação Canônica de Dados

Até então nos referimos a *dados* sem nos preocupar com sua representação. Infelizmente, representar dados não é tarefa trivial, posto que diferentes computadores adotam representações diferentes para o mesmo dado. Por exemplo, a IBM de longa data emprega o código EBCDIC em seus *mainframes* para representar texto. Os demais fabricantes empregam o código ASCII. Outro problema é a sequência de bits em uma palavra: alguns computadores empregam o bit mais significativo à esquerda, enquanto outros à direita. Por exemplo, números inteiros são representados de maneira distinta nas arquiteturas Sparc e Pentium. Como regra geral, a forma de representação interna de dados adotada por computadores é fortemente influenciada por suas arquiteturas de hardware.

Com o advento das redes heterogêneas (interconectando hardware de diferentes arquiteturas), tornou-se necessário uma representação canônica de dados (ou sintaxe de transferência). A idéia é simples. Antes de transmitir um dado, o host converte de sua representação interna para a representação canônica. Ao recebê-lo, o host receptor converte da representação canônica para sua representação interna. Em outras palavras, os dados

que fluem pela rede têm uma representação padronizada. Isto evita que hosts comunicantes necessitem conhecer mutuamente a forma de representação de dados por eles empregadas ¹.

No modelo OSI, a representação canônica de dados baseia-se em uma sintaxe denominada ASN.1 (Abstract Syntax Notation, version 1).

7.2 A Sintaxe ASN.1

ASN.1 é uma linguagem formal de especificação de tipos de dados padronizada pela ISO (documento 8824). Uma segunda padronização, BER (Basic Encoding Rules), especifica como dados na sintaxe ASN.1 são formatados em uma mensagem (por exemplo, um TPDU). BER é também um padrão ISO (documento 8825).

Aplicações definem todas as estruturas de mensagens válidas (tipos de PDUs) em ASN.1, agrupando estas definições em um único módulo: o *dicionário de tipos*. Ao transmitir um PDU, a aplicação passa à camada de apresentação o tipo de PDU sendo transmitido e o conteúdo de seus campos. A camada de apresentação consulta o dicionário de tipos em busca da definição dos campos do PDU e, utilizando a norma BER, gera a mensagem a ser transmitida. A figura 7.1 ilustra este processo.

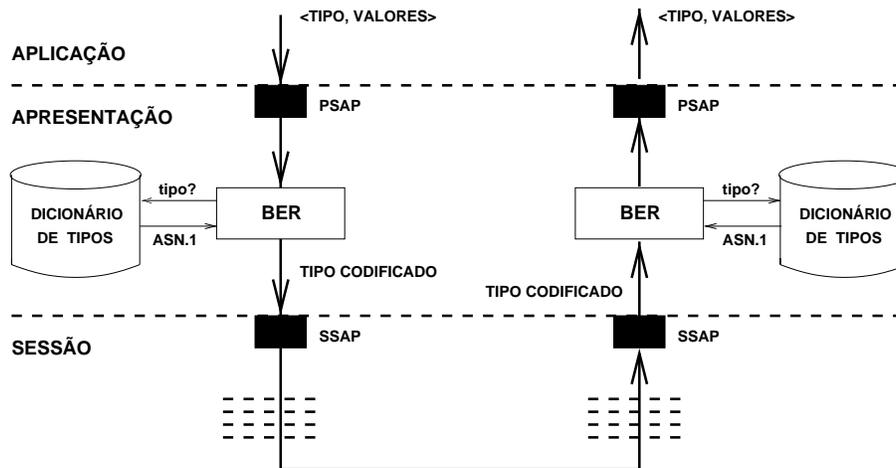


Figura 7.1: O serviço OSI de apresentação.

Quando a mensagem atingir a camada de apresentação do receptor, é verificado o tipo de PDU. O dicionário de tipos é então consultado para se obter a estrutura do PDU recebido. Com base sua na estrutura e, sabendo-se que a codificação segue ASN.1/BER, o receptor pode remontar o PDU segundo sua representação própria de dados.

Como toda a linguagem formal, ASN.1 consiste em um conjunto de elementos básicos (*tokens*) e regras de combinação destes elementos. ASN.1 define quatro tipos de *tokens*:

¹Esta solução é inviável por três razões: a quantidade de arquiteturas (e portanto representações) distintas; o aparecimento contínuo de novas arquiteturas; e a impossibilidade de difusão de mensagens em um formato inteligível por todos os hosts.

palavras: sequências de letras maiúsculas ou minúsculas, dígitos e hífen. O primeiro caractere deve ser uma letra. Palavras definem tipos de dados, identificadores e palavras-chave.

números: formações compostas apenas de dígitos.

strings: cadeia de caracteres do tipo

- alfanumérico: qualquer conteúdo delimitado por
- hexadecimal: conteúdo entre 0-9 e A-F delimitado por 'H
- binário: 0 ou 1 delimitados por 'B

pontuação: caracteres do tipo

. : = , { } < > - ' " |

7.2.1 Tipos Primitivos de Dados

ASN.1 define um conjunto de tipos primitivos de dados. Tipos mais complexos são obtidos através da agregação dos tipos primitivos. Os tipos primitivos são:

boolean: valores lógicos (TRUE-FALSE);

integer: números inteiros;

real: números reais (mantissa, base, expoente);

bit string: sequência ordenada de 0s e 1s;

octet string: sequência ordenada de bytes (octetos);

any: tipo a ser especificado;

null: tipo sem valor;

object descriptor: referencia algum objeto sem sintaxe formal (usado mais a título de comentário);

object identifier: define univocamente um objeto, por exemplo, um protocolo de transferência de arquivos;

encrypted: tipo criptografado.

Exemplos:

```
ChecksumInUse ::= BOOLEAN
checksum ChecksumInUse ::= FALSE

TimeToLive ::= INTEGER
time TimeToLive ::= 60

Status ::= INTEGER { up(1), down(2), testing(3) }
status Status = up

Option ::= INTEGER (0..5)
valor Option ::= 2

Timeout ::= REAL
timeout Timeout ::= {250, 10, -3}

Preamble ::= BITSTRING
preambulo Preamble ::= '01010101'B

IpAddress ::= OCTET STRING (SIZE (4))
ipaddr IpAddress ::= '8F6A3290'H -- 143.106.50.144

ForFutureUse ::= ANY

NoLongerInUse ::= NULL

Ftam ::= OBJECT DESCRIPTOR
ftan Ftam ::= "File Transfer, Access and Management"

Ftam ::= OBJECT IDENTIFIER
ftan Ftam ::= { iso standard 8571 }
```

7.2.2 Tipos Complexos (Construtores)

Tipos construtores são formados a partir dos tipos primitivos descritos na seção anterior. São eles:

- sequence:** lista ordenada de tipos primitivos arbitrários;
- sequence of:** lista ordenada de tipos primitivos homogêneos;
- set:** lista não ordenada de tipos primitivos arbitrários;
- set of:** lista não ordenada de tipos primitivos homogêneos;

choice: um conjunto de tipos dos quais apenas um deve ser considerado.

Exemplos:

```
PDUHeader ::= SEQUENCE {
    Length INTEGER,
    Flags BITSTRING,
    DestinationTSAP INTEGER,
    SourceTSAP INTEGER }
```

```
ProtocolClassOption ::= CHOICE {
    class-0 OCTETSTRING,
    class-1 OCTETSTRING,
    class-2 OCTETSTRING,
    class-3 OCTETSTRING,
    class-4 OCTETSTRING }
```

7.2.3 Tipos Marcados (*Tagged*)

Tipos marcados carregam uma informação adicional para fins de eliminação de ambiguidades. Vimos pelos exemplos acima que tipos construtores são estruturas compostas de múltiplos campos. Durante a transmissão de um tipo construtor, o tipo tamanho e valor de cada um de seus campos é codificado segundo a norma BER. Um problema decorre da existência de campos opcionais. A palavra-chave `OPTIONAL` pode ser usada para identificar um campo como opcional. Outra palavra-chave, `DEFAULT`, estabelece o valor que deve ser tomado caso o campo (opcional) seja omitido. Por exemplo:

```
PDUHeader ::= SEQUENCE {
    Length INTEGER,
    Flags BITSTRING OPTIONAL DEFAULT '00000000'B,
    DestinationTSAP INTEGER,
    SourceTSAP INTEGER
}
```

Ao receber um PDU com o cabeçalho dado pela sequência acima, como o receptor reconhece se o campo opcional foi transmitido? A solução é marcar explicitamente os componentes da sequência e transmitir a marca junto com o dado. Exemplo:

```
PDUHeader ::= SEQUENCE {
    Length [0] INTEGER,
    Flags [1] BITSTRING OPTIONAL DEFAULT '00000000'B,
    DestinationTSAP [2] INTEGER,
```

```
SourceTSAP [3] INTEGER
}
```

Assim sendo, se o campo *Flags* for omitido, o receptor toma conhecimento pela ausência da marca [1]. Com a transmissão da marca, o tipo fica redundante pois pode ser acessado pelo receptor na definição do tipo construtor. Esta redundância pode ser mantida como uma verificação adicional. Para eliminá-la deve-se utilizar a palavra-chave IMPLICIT:

```
PDUHeader ::= SEQUENCE {
  Length [0] IMPLICIT INTEGER,
  Flags [1] IMPLICIT BITSTRING OPTIONAL DEFAULT '00000000'B,
  DestinationTSAP [2] IMPLICIT INTEGER,
  SourceTSAP [3] IMPLICIT INTEGER
}
```

Suponha agora que o protocolo possui vários tipos de PDUs (por exemplo, 9 para o TP-4). Ao receber um PDU, como o receptor descobre de que tipo se trata? Se o cabeçalho do exemplo anterior for, digamos, do terceiro tipo, podemos marcar não apenas seus campos, mas também sua estrutura como um todo:

```
PDUHeader ::= [APPLICATION 3] SEQUENCE {
  Length [0] IMPLICIT INTEGER,
  Flags [1] IMPLICIT BITSTRING OPTIONAL DEFAULT '00000000'B,
  DestinationTSAP [2] IMPLICIT INTEGER,
  SourceTSAP [3] IMPLICIT INTEGER
}
```

A palavra-chave APPLICATION denota um tipo construtor no dicionário de tipos da aplicação OSI. A marca, 3, indica se tratar da terceira definição presente no dicionário. APPLICATION pode ser substituída por PRIVATE (aplicação privada), CONTEXT-SPECIFIC (interpretação dependente do contexto) e UNIVERSAL (tipos primitivos do ASN.1).

7.2.4 Macros

Macro é um recurso utilizado em muitas linguagens de programação para redefinir a sintaxe da linguagem. ANS.1 permite a definição de macros através da construção

```
<<name-of-macro>> MACRO ::=
BEGIN
```

```
TYPE NOTATION ::= <<types>>
```

```
VALUE NOTATION ::= <<value>>
```

```
Productions
```

```
END
```

TYPE NOTATION contém os tipos (atributos) definidos pela macro, enquanto VALUE NOTATION determina o tipo do objeto que deve ser passado quando a macro é invocada. Productions (produções) são similares à notação BNF utilizada na descrição formal de linguagens de programação e impõem regras para a construção de atributos.

Macros ANS.1 são utilizadas para a definição de várias construções na área de gerência de redes tais como de identificação de módulos e tipos de objetos de gerência.

7.3 Sintaxe de Transferência

Sintaxe de transferência especifica como um tipo ASN.1 é formatado em uma mensagem. A norma BER fornece regras para esta formatação. Dada a sua extensão, veremos apenas os aspectos fundamentais do BER.

BER adota inteiramente o código ASCII para caracteres. As representações adotam o bit mais significativo a esquerda. Números são representados pelos seus complementos de dois.

Tipos primitivos ou não são formatados em três campos: a identificação do tipo, o tamanho do dado, e o valor propriamente dito. Caso o tipo não seja primitivo, o valor irá conter também outros três campos (tipo, tamanho, valor), e assim sucessivamente até chegarmos unicamente a tipos primitivos.

7.3.1 Identificação do Tipo, Tamanho e Valor

O tipo é identificado por um byte. Os primeiros dois bits informam a classe do tipo (UNIVERSAL, APPLICATION, CONTEXT-SPECIFIC e PRIVATE). O próximo bit denota se o tipo é primitivo ou não. Os cinco bits restantes identificam o tipo na classe. Para a classe UNIVERSAL, 1 denota BOOLEAN, 2 denota INTEGER, 3 denota BITSTRING, 4 denota OCTETSTRING, e assim por diante.

Caso uma classe tenha mais que 30 tipos, ativa-se todos os 5 bits do tipo seguindo-se tantos bytes quantos forem necessários para abrigar o índice máximo, todos começando com 1 no primeiro bit (restando sete para o tipo), exceto o último.

O tamanho do dado, se menor que 128 ocupa um único byte, tendo o primeiro bit com valor 0. Se maior que 128, o primeiro bit é feito 1 e os sete restantes estipulam quantos bytes a seguir armazenam o valor. Um tamanho de conteúdo nulo (todos os bits 0) indica que o valor será delimitado por dois bytes de conteúdo nulo em cada extremo.

A codificação de valores varia com o tipo. Por exemplo, um OCTETSTRING de N bytes é codificado empregando-se um byte por caractere, segundo o código ASCII. Exemplos:

```

numero 100:      00000010 00000001 01100100

numero 32768:   00000010 00000010 01000000 00000000

string "A":     00000100 00000001 01100001

```

Como exemplo de tipos construtores, tomemos o caso de uma SEQUENCE. Neste caso, o byte que descreve o tipo indica se tratar de uma sequência, o byte tamanho indica o número de elementos da sequência (N) e o byte valor dá lugar a N triplas (tipo, tamanho, valor). Por exemplo, uma sequência composta do número 100 e do string "A" é codificada como:

```

Exemplo ::= [UNIVERSAL 7] SEQUENCE {
            elem-1 [1] IMPLICIT INTEGER,
            elem-2 [2] IMPLICIT OCTETSTRING }

```

Para valores <100, "A">, a sequência acima é codificada como:

```

01100111 00000001 00000010  <-  [UNIVERSAL 7] 2
00000001 01100001 01100100  <-  [1] 100
00000010 01100001 01100001  <-  [2] "A"

```

7.4 Primitivas OSI de Apresentação

As primitivas OSI para a camada de apresentação são as mesmas, exceto uma, da camada de sessão ² (ver figura 6.1). Simplesmente tais primitivas “curto-circuitam” a camada de apresentação, permitindo à camada de aplicação o acesso às funções da camada de sessão. Tais primitivas existem para manter o conceito de camadas.

Uma nova primitiva S-ALTER-CONTEXT provê um mecanismo de mudança de contexto, isto é, passar de uma atividade de sessão para outra sem suspender a primeira. Esta primitiva é um serviço com confirmação.

7.5 Problemas

1. Quais as funções da camada de apresentação?
2. Especifique os NPDUs do X.25/camada 3 em ASN.1.

²Começando com P- ao invés de S-

Capítulo 8

A CAMADA DE APLICAÇÃO

8.1 Funcionalidades da Camada de Aplicação

A camada de aplicação do modelo OSI define protocolos (e entidades que os implementam) utilizados por aplicativos denominados APs (Application Processes ou Processos de Aplicação). Por exemplo, um AP que implementa um serviço de mensagens (via caixa postal, por exemplo) quando operando em um ambiente OSI utilizaria o protocolo MHS (Message Handling System) definido na camada de aplicação.

A camada de aplicação tem por finalidade prover os seguintes serviços aos APs que dela se utilizam:

- identificar elementos remotos de aplicação, verificando sua disponibilidades (servidores, por exemplo);
- negociar com estes elementos uma qualidade de serviço apropriada;
- negociar contextos de apresentação para troca de informação;
- negociar serviços de sessão;
- transportar dados entre aplicativos;
- autenticar as entidades comunicantes.

Alguns exemplos de serviços providos pela camada de aplicação são listados abaixo:

- serviço de acesso e transferência de arquivos;
- serviço de troca de mensagens;
- serviço de diretório;
- serviço de manipulação de tarefas (jobs) remotas;
- serviço de controle de concorrência e recuperação;

- serviço de terminal remoto;
- etc.

Para o provimento destes serviços a camada de aplicação possui uma estruturação relativamente complexa quando comparada às demais camadas do modelo OSI. Esta estruturação é sintetizada na sequência.

8.2 Estruturação da Camada de Aplicação

A camada de aplicação agrega *entidades de aplicação* responsáveis pelo provimento de serviços aos APs. Estes serviços podem ser de propósito geral ou específico. Os serviços de propósito geral são providos por entidades denominadas CASE (Common Application Service Elements), enquanto os de propósito específicos são providos pelas entidades SASE (Specific Application Service Elements). Em linhas gerais, APs usam serviços oferecidos por SASEs que por sua vez usam serviços oferecidos pelos CASEs conforme ilustrado na figura 8.1.

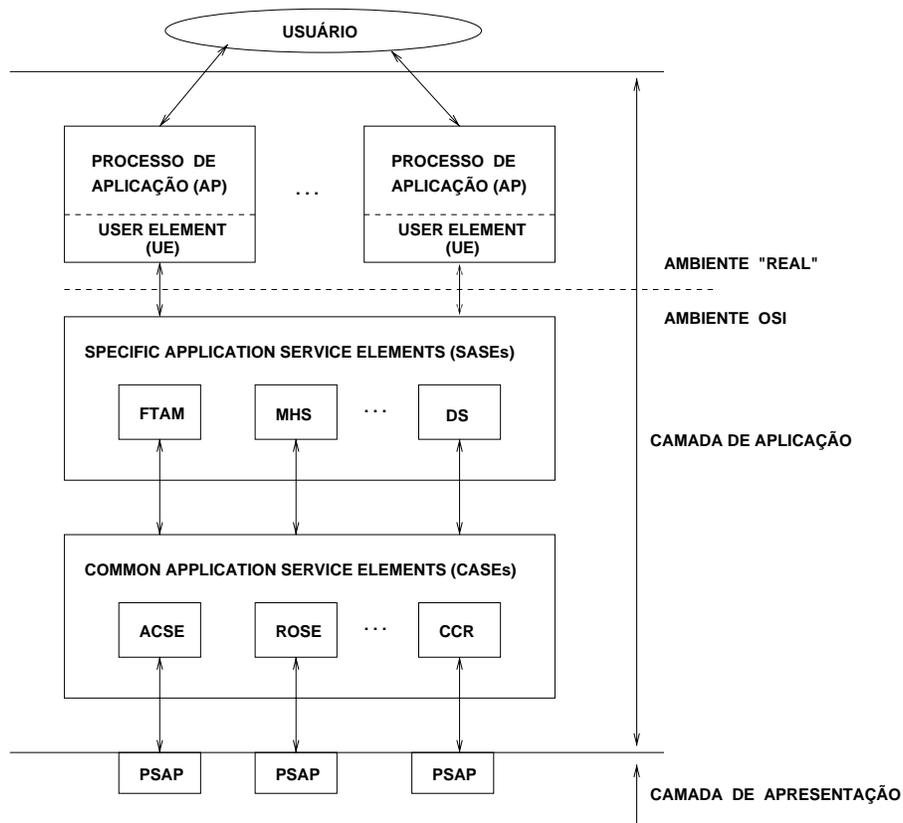


Figura 8.1: Os componentes da camada de aplicação do modelo OSI.

Um conceito importante na camada de aplicação é o conceito de *associação*. As demais camadas do modelo OSI oferecem a abstração de conexão (primitivas tipo CONNECT) como

o meio interação entre duas entidades da camada $N+1$ via serviços oferecidos pela camada N . O que ocorre se $N = 7$, ou, em outras palavras, *quem* utilizaria as conexões da camada de aplicação?

Este problema foi resolvido através das regras abaixo:

1. a camada de aplicação define associação (não conexão) como forma de provimento de serviços aos APs;
2. cada associação de aplicação utiliza uma única conexão de apresentação (isto é, uma associação pode ser vista como uma extensão de uma conexão de apresentação).

Um CASE denominado ACSE (Association Control Service Element) oferece o serviço de estabelecimento de associação para as demais entidades da camada de aplicação.

Atualmente, a tendência é evitar a distinção entre CASE e SASE e denominar estas entidades simplesmente de ASE (Application Service Element). Uma questão crucial é: como combinar ASEs para implementar um serviço complexo? Esta questão está vinculada com o conceito de associação. Intuitivamente, um ASE proveria uma associação através de uma conexão de apresentação agregando valor à conexão oferecida pela camada inferior¹. Este modelo apresenta alguns problemas quando ASEs são combinados. O principal problema refere-se à complexidade da coordenação de múltiplas associações (cada qual utilizando uma única conexão de apresentação). O que ocorre, por exemplo, se uma das associações é encerrada pelo provedor?; ou, como gerenciar o diálogo a nível de aplicação se diferentes associações utilizam diferentes serviços de sessão²?

A proposta da ISO para esta questão é a definição de um componente agregador denominado ASO (Application Service Object). Um ASO é um objeto provedor de serviço que agrega:

- Elementos de Serviço de Aplicação (ASE);
- Outros ASOs;
- Uma única Função de Controle (CF: Control Function) que especifica como os componentes de um ASO são agregados.

Um ASO estabelece uma única associação eliminando, portanto, o problema de gerenciamento de múltiplas associações (uma por ASE). Obviamente, caso um ASO agregue outros ASOs múltiplas associações emanarão deste, mas as demais associações são encapsuladas pelos ASOs agregados.

Entidades de aplicação agora são definidas em termos de ASOs, não mais em termos de ASEs. Em linhas gerais, neste modelo os ASEs definem mensagens (APDUs), enquanto os ASOs (via CF) determinam a dinâmica da interação através destas mensagens.

A figura 8.2 ilustra esta nova concepção da camada de aplicação. Infelizmente, esta concepção foi elaborada após muitos ASEs já terem sido padronizados. Esta padronização não levou em conta a possibilidade destes ASEs serem agregados em torno de um ASO. A

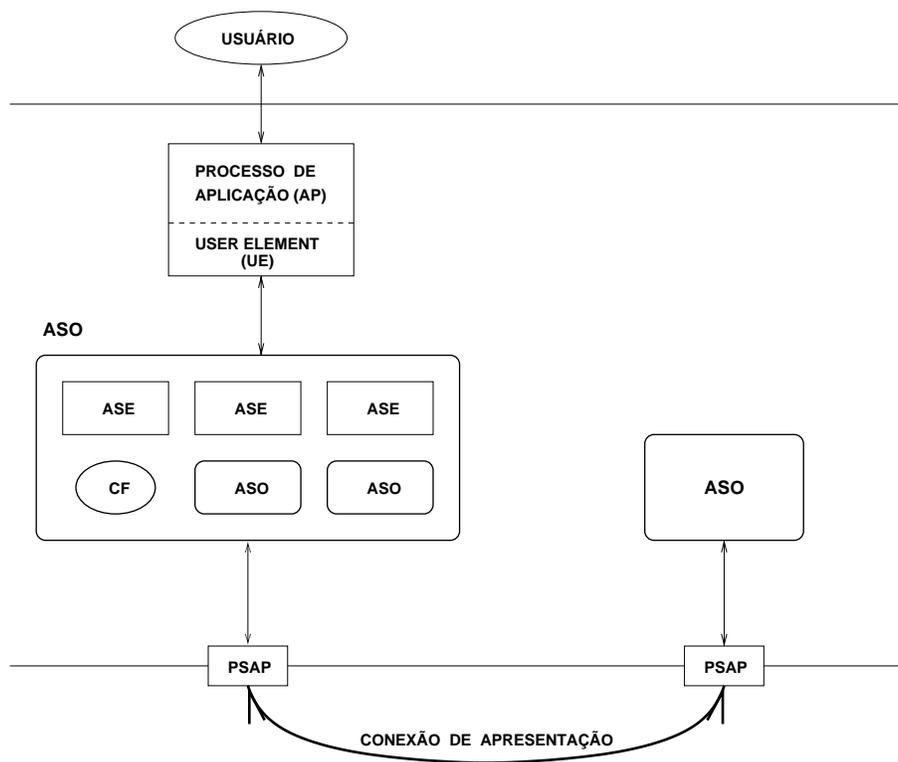


Figura 8.2: Camada de aplicação baseada em ASOs.

reformulação destes padrões para adequarem os ASEs à nova estruturação da camada de aplicação é um processo incerto.

A seguir apresentaremos alguns elementos de serviço da camada de aplicação.

8.3 Serviço de Associação (ACSE)

Conforme mencionado anteriormente, o elemento de aplicação responsável pelo estabelecimento de associação é o ACSE. O ACSE utiliza uma conexão de apresentação para cada associação que estabelece. Um dado interessante é que após estabelecida a associação, a entidade que a requisitou passa a utilizar diretamente a conexão de apresentação aberta para esta associação para a troca de dados. Entretanto, o encerramento da associação se dá através do ACSE.

Além dos parâmetros requeridos pela primitiva P-CONNECT, o estabelecimento de uma associação necessita:

- identificadores das entidades de aplicação envolvidas na associação (iniciadora e respondedora);

¹As demais camadas operam segundo este princípio.

²Por exemplo, uma utiliza *tokens* para gerenciar o diálogo e a outra não.

- um contexto de aplicação: um identificador de objeto em ASN.1 utilizado para identificar as regras de intercâmbio de informação através da associação (por exemplo, protocolo de aplicação);
- dados supridos pela entidade usuária (não interpretados pelo ACSE).
- um contexto de autenticação: um identificador de objeto em ASN.1 utilizado para identificar as regras de autenticação (por exemplo, chaves criptográficas).

Os serviços providos pelo ACSE são acessados através das primitivas abaixo:

1. A-ASSOCIATE: estabelece uma associação utilizando a primitiva P-CONNECT da camada de apresentação. Este serviço é confirmado.
2. A-RELEASE: termina uma associação utilizando a primitiva P-RELEASE de apresentação. Este serviço é confirmado.
3. A-ABORT: termina abruptamente (aborta) uma associação por solicitação da entidade usuária utilizando a primitiva P-U-ABORT de apresentação. Este serviço é sem confirmação.
4. A-P-ABORT: informa o término abrupto de uma associação pelo provedor do serviço. Esta primitiva é ativada quando da ocorrência de um P-P-ABORT na camada de apresentação e possui o modo indicação apenas.
5. A-UNITDATA: transfere uma unidade de dados de aplicação sem o estabelecimento de associação. Este serviço é sem confirmação e utiliza a primitiva P-UNITDATA de apresentação.

Note a inexistência da primitiva A-DATA para envio de dados através de uma associação. A primitiva P-DATA é empregada para esta finalidade.

Um ponto importante do ACSE é como entidades de aplicação são identificadas. A identificação AEs se dá através de nomes que o ACSE mapeia em endereços de apresentação (PSAPs) utilizando o próprio serviço de diretório da camada de aplicação.

8.4 Serviço de Operações Remotas (ROSE)

ROSE (Remote Operations Service Element) é um ASE que intermedia a submissão remota de operações. Exemplo de tais operações são: armazenamento de mensagens e submissão de tarefas (jobs) remotas.

O ROSE fornece um serviço similar à uma chamada de procedimento remoto (RPC: Remote Procedure Call), sendo que este serviço pode ter natureza síncrona ou assíncrona. No modo síncrono a entidade solicitante permanece bloqueada até a entidade executora retornar o resultado da operação. No modo assíncrono, a entidade solicitante pode evocar múltiplas operações e só então coletar os respectivos resultados. Sob esta ótica, as operações remotas são agrupadas em cinco classes:

- classe 1: operação síncrona que retorna indicativo de sucesso ou falha;
- classe 2: operação assíncrona que retorna indicativo de sucesso ou falha;
- classe 3: operação assíncrona que retorna somente indicativo de falha;
- classe 4: operação assíncrona que retorna somente indicativo de sucesso;
- classe 5: operação assíncrona que não retorna qualquer indicativo.

O ROSE utiliza associações para a comunicação entre entidades remotas³. Uma associação aberta para suporte à operações remotas define através de sua classe qual entidade está autorizada a evocar procedimentos remotos:

- classe 1: permite apenas a entidade que solicitou a associação evocar operações remotas;
- classe 2: permite apenas a entidade que aceitou a associação evocar operações remotas;
- classe 3: permite ambas as entidades evocarem operações remotas.

O ROSE provê seus serviços através de cinco primitivas:

1. RO-INVOKE: solicita a execução de uma operação remota;
2. RO-RESULT: retorna o resultado da operação remota;
3. RO-ERROR: reporta um erro na execução de uma operação remota;
4. RO-REJECT-U: rejeita uma solicitação ou retorno se a entidade usuária detectar um erro (por exemplo, operação inexistente ou falha de autenticação);
5. RO-REJECT-P: rejeita uma solicitação ou retorno se o provedor de serviço detectar um erro.

Todos os serviços são sem confirmação (RO-REJECT-P possui o modo indicação apenas).

O ROSE define quatro classes de PDUs, um para cada tipo de primitiva. Argumentos, resultados e erros das operações são especificados em ASN.1.

O ROSE provê ainda o conceito de operações ligadas (linked). Neste mecanismo um argumento de uma operação identifica outra operação dirigida para a própria entidade solicitante. Um dos objetivos é evitar a passagem de argumentos volumosos (matrizes, por exemplo) através da execução de operações no local onde os argumentos estão armazenados (emulando a passagem de argumentos por referência). A figura 8.3 ilustra este exemplo.

³Propostas recentes permitem operações remotas sem o estabelecimento de associação através da primitiva A-UNITDATA.

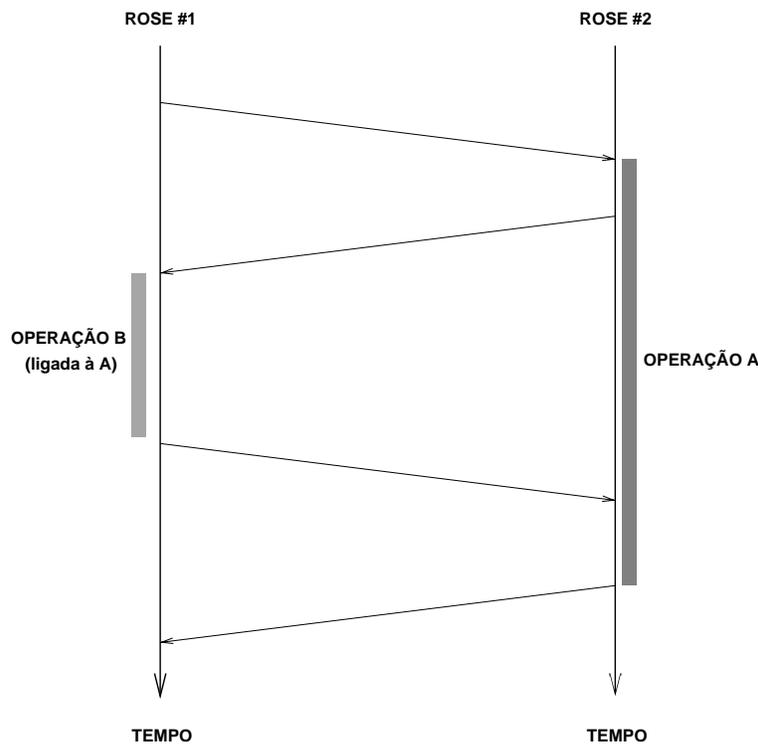


Figura 8.3: Operações ligadas no ROSE.

8.5 Serviço de Transferência Confiável (RTSE)

RTSE (Reliable Transfer Service Element) é um ASE que provê a transferência confiável de dados entre duas entidades de aplicação. Por confiável subentende-se a capacidade de recuperação de erros (quebras de conexões, panes de hardware, etc.).

O RTSE fornece uma interface mais simples que a interface da camada de sessão, além de “esconder” a existência do ACSE. Por exemplo, o RTSE insere automaticamente pontos de sincronismo no fluxo de dados sendo transferido, além de gerenciar a atividade de sessão aberta para a transferência. O RTSE é utilizado, por exemplo, nas implementações MHS (Message Handling System) para transferência confiável de mensagens entre comutadores (relays).

O RTSE emprega transferência alternada nos dois sentidos (TWA: Two Way Alternate), com mecanismo de *token* para gerenciamento do sentido da comunicação.

Para cada PDU submetido para transferência, o RTSE abre uma nova atividade no nível de sessão. O PDU é então segmentado em blocos, sendo a transferência de cada bloco precedida por um ponto de sincronismo menor. Quando todos os blocos tiverem sua recepção confirmada o RTSE confirma à entidade usuária a entrega do PDU e encerra a atividade. Caso um bloco não tenha sido confirmado, o RTSE solicita ao outro extremo uma retroação ao ponto de sincronismo inserido anteriormente à este bloco.

Um ponto importante do RTSE é que este assume que os dados sendo transferido são constituídos de *string* de bytes, isto é, o tipo ASN.1 OCTET STRING. A razão desta imposição é facilitar a inserção automática de pontos de sincronismo menor para fins de retroação ante

a ocorrência de falhas. Isto sem dúvida consiste em uma limitação severa posto que as entidades usuárias do RTSE certamente desejariam transferir dados bem mais estruturados. Para contornar esta limitação, a especificação RTSE define um *serviço de compatibilização sintática*⁴. Este serviço permite que as entidades usuárias definam implementações locais que transformam os tipos de dados peculiares da entidade em *string* de octetos. Regras para esta transformação está fora da especificação do RTSE.

O RTSE dispõe de sete primitivas:

- RT-OPEN: solicita o estabelecimento de uma associação. É um serviço confirmado.
- RT-CLOSE: solicita o encerramento de uma associação. É um serviço confirmado.
- RT-TRANSFER: transfere dados. É um serviço confirmado.
- RT-TURN-PLEASE: solicita à outra entidade habilitação para transferir dados. É um serviço não confirmado.
- RT-TURN-GIVE: passa o *token* à outra entidade, habilitando-a a transferir dados. É um serviço não confirmado.
- RT-U-ABORT: termina abruptamente uma associação. É um serviço não confirmado.
- RT-P-ABORT: o provedor sinaliza a impossibilidade de manter a associação. Possui o modo indicação apenas.

8.6 Serviço de Controle de Concorrência e Recuperação (CCR)

CCR (Commitment, Concurrency and Recovery) é um serviço de suporte a transações atômicas. Uma transação atômica é um conjunto de operações (tipicamente sobre uma massa de dados) cuja execução deve satisfazer a três propriedades:

1. atomicidade: ou todas as operações se completam com sucesso, ou tudo que a transação produziu é desfeito;
2. serialização: as operações pertencentes a transações atômicas distintas não se entrelaçam;
3. persistência: quando uma transação atômica termina com sucesso, o resultado de suas operações não se perde ante a falhas não catastróficas tais como queda de hosts e rede.

Transações atômicas têm por finalidade manter consistente o estado de determinados objetos (massas de dados, programas, etc) quando manipulados por operações sujeitas a falhas.

⁴Syntax Matching Service.

Uma transação atômica opera sobre objetos localizados em um ou mais hosts. No último caso, um host é eleito como *coordenador* da transação (geralmente o host da aplicação que solicitou a transação) sendo os demais *participantes*. Tanto o coordenador quanto os participantes são responsáveis pela garantia das três propriedades acima. Um instante crítico para esta garantia é o encerramento da transação. Um protocolo de consenso se faz necessário e o mais utilizado é o protocolo de *compromissamento em duas fases*⁵.

A implementação de transações atômicas não é tarefa trivial. Mecanismos de recuperação (*rollback*) são necessários para a garantia da atomicidade; de bloqueio (*locking*) para garantir a serialização; e de *logging* para a garantia da persistência.

O CCR não oferece tais mecanismos, posto que são fortemente dependentes da aplicação. Em resumo, o CCR provê primitivas para iniciar, abortar e terminar uma transação segundo o protocolo de compromissamento em duas fases.

As primitivas CCR são:

- C-BEGIN: o coordenador informa aos participantes o início de uma transação atômica (sem confirmação);
- C-PREPARE: o coordenador solicita aos participantes o encerramento da transação (sem confirmação);
- C-READY: o participante responde o coordenador que está pronto para encerrar a transação (sem confirmação). Neste momento, todas as operações efetuadas pela transação são feitas permanentes (gravadas em disco).
- C-REFUSE: o participante responde o coordenador que está impossibilitado de encerrar a transação com sucesso ⁶ (sem confirmação);
- C-COMMIT: o coordenador informa aos participantes que a transação pode se encerrar com sucesso (caso todos os participantes tenham respondido C-READY). Este serviço é confirmado;
- C-ROLLBACK: o coordenador informa aos participantes que a transação deve ser abortada (caso pelo menos um participante respondeu C-REFUSE). Coordenador e participantes desfazem todas as operações efetuadas pela transação. Este serviço é confirmado;
- C-RESTART: reinicia, se possível, uma transação a partir de um ponto especificado. Este serviço é confirmado.

Note a inexistência de uma primitiva tipo C-DATA. Tal qual o RTSE, o CCR utiliza os serviços da camada de apresentação para transferência de dados. O CCR fornece apenas os mecanismos de controle para o processamento de transações. As operações realizadas no âmbito de uma transação estão fora da especificação CCR. Estas podem se basear no ROSE ou simplesmente utilizar a primitiva P-DATA de apresentação para o envio de informação.

⁵*Two-phase commit.*

⁶Provavelmente porque alguma de suas operações falhou, ou está impossibilitado de tornar as operações permanentes.

Do ponto de vista da implementação, uma transação é delimitada por pontos de sincronismo maior. As primitivas C-REFUSE, C-ROLLBACK e C-RESTART emitem um P-RESYNCHRONIZE a fim de retroagir (via facilidades de *logging*) ao ponto de sincronismo inserido no início da transação.

Uma especificação correlata denominada TP (Transaction Processing) incorpora todas as primitivas do CCR, definindo inclusive uma primitiva TP-DATA.

8.7 Serviço de Troca de Mensagens (MHS)

MHS (Message Handling System) é um serviço de manipulação⁷ de mensagens. Este serviço é conhecido como X.400 por ter se originado na CCITT e publicado com esta identificação.

MHS se destina tanto ao suporte de sistemas de mensagens inter-pessoal (correio eletrônico) quanto documentos comerciais (EDI: Electronic Data Exchange). O MHS não se restringe a caracteres ASCII. Mensagens compostas de textos, imagens, gráficos, etc., podem ser manipuladas pelo MHS.

O modelo MHS consiste de vários elementos funcionais, descritos a seguir. Um Agente do Usuário (UA: User Agent) pode ser visto como um aplicativo que interage com o usuário. Por usuário subentende-se uma pessoa ou um programa que faz uso do MHS. O UA é encarregado de preparar, enviar e receber mensagens.

O envio de mensagens se dá através de Agentes de Transferência de Mensagens (MTA: Message Transfer Agent). MTAs são comutadores de mensagens, isto é, armazenam temporariamente e enviam a outro MTA (política “Armazena e Envia”) até ser entregue ao UA servindo o destinatário. Note que não é aberta conexão direta entre o originador e o destinatário da mensagem; ao contrário, a mensagem é “chaveada” entre MTAs analogamente à comutação de pacotes. MTAs cooperam segundo um protocolo denominado P1 e utilizam o RTSE para a transferência confiável da mensagem entre MTAs. Acesso aos MTAs se dá através de um protocolo denominado P3.

MS (Message Store) é um elemento funcional que atua entre o UA e o MTS. Sua função é armazenar temporariamente a mensagem até ser entregue ao MTS (no caso de envio) ou ao UA (no caso de recepção). Operações de armazenamento e recuperação nos MSs são executadas segundo um protocolo denominado P7. Tais operações são executadas remotamente nos MSs através dos serviços providos pelo ROSE.

MTS (Message Transfer System) é uma coleção de MTAs que atuam cooperativamente no roteamento de mensagens. É análogo a uma rede de comutação de pacotes (onde os MTAs seriam os roteadores).

A figura 8.4 ilustra a interligação dos elementos funcionais descritos acima.

Uma mensagem X.400 é composta de um *envelope* e um *conteúdo* conforme ilustrado na figura 8.5.

O endereçamento X.400 é utilizado tanto para identificar originador e destinatário(s) quanto para rotear as mensagens. Um endereço X.400 é composto dos seguintes campos:

- Nome de País: código de 2 (ISO 3166) ou 3 (CCITT X.121) caracteres indicando o país do usuário (Ex: BR);

⁷Endereçamento, armazenamento, envio, notificação, etc.

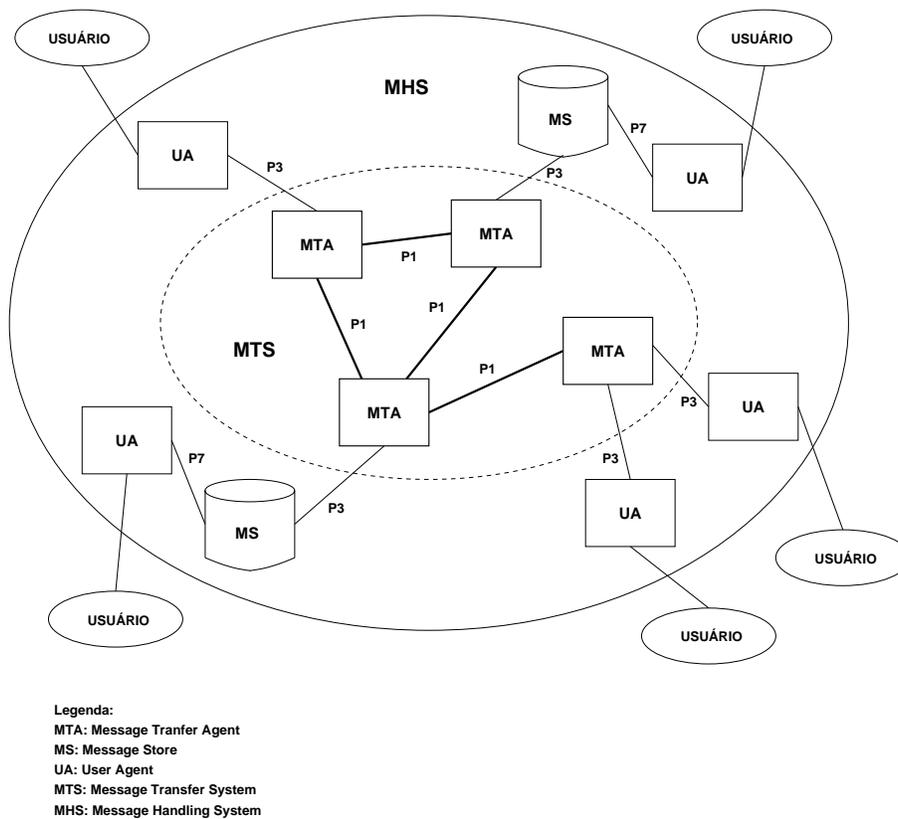


Figura 8.4: Elementos funcionais do MHS.

- Nome ADMD (Administrative Management Domain): especifica a organização provedora do serviço ao qual o usuário está conectado (Ex: Embratel);
- Nome PDMD (Private Management Domain): eventual corporação que intermedia o oferecimento do serviço (ex: um provedor de acesso);
- Nome da Organização à qual o usuário pertence;
- Nome da Unidade Organizacional à qual o usuário pertence;
- Nome Pessoal do usuário;
- Atributos próprios do domínio;
- Endereço X.121 do terminal do usuário.

Nem todos os atributos acima podem estar presentes no envelope de uma mensagem. Por exemplo, o endereço do terminal pode suprimir o nome da organização e usuário.

Quanto a segurança e privacidade, o protocolo X.400 cobre três áreas:

1. confidencialidade: garante que apenas o destinatário terá acesso ao conteúdo da mensagem;

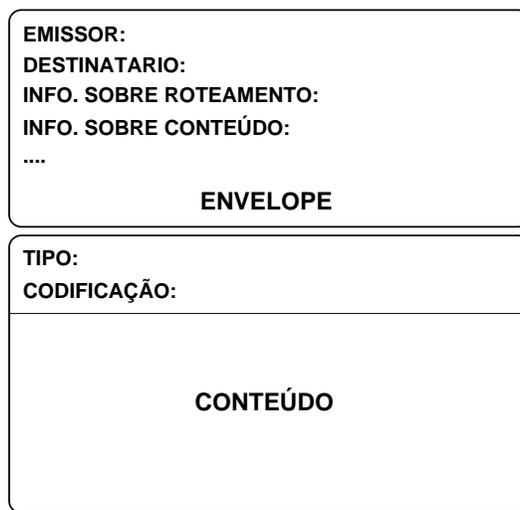


Figura 8.5: Estrutura de mensagens no MHS.

2. autenticação: garante a identidade do originador⁸;
3. provas de submissão e entrega: notificações que comprovam o envio/recepção.

8.8 Serviço de Diretório (DS)

DS (Directory Service) é um serviço de base de dados global (DIB: Directory Information Base). Esta base de dados forma um “diretório eletrônico” que armazena objetos tais como pessoas, endereços, senhas, organizações, etc. Este serviço é parte das recomendações da série X.500 da ex-CCITT.

Cada objeto do diretório possui um ou mais *nomes distintos* que servem como chave de acesso para este objeto. Nomes distintos são coleções ordenadas de atributos tais como:

- cn: common name (nome usual);
- ou: organizational unit (unidade organizacional);
- o: organização;
- st: state (estado);
- c: country (país).

Um objeto é uma sequência de atributos especificado em ASN.1 onde cada atributo possui um identificador de objeto que descreve o tipo de atributo e o(s) correspondente(s) valor(es). Tipos usualmente são *PrintableString* em ASN.1 o que possibilita uma interpretação livre de arquitetura de máquina. A figura 8.6 ilustra um objeto genérico.

⁸Isto é, impede que uma mensagem seja enviada em nome de outrém.

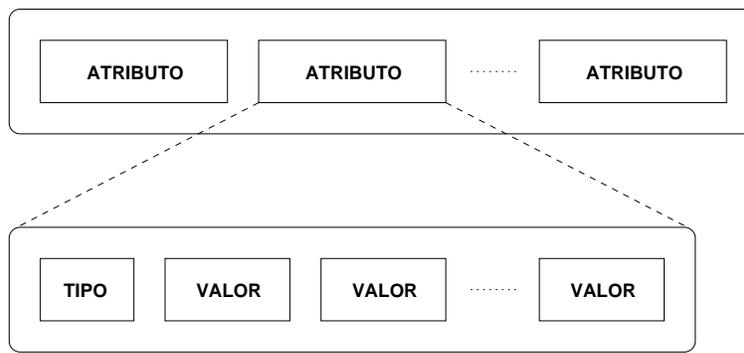


Figura 8.6: Modelo de objeto para o Serviço de Diretório.

Objetos são agrupados em classes, sendo algumas classes padronizadas pela recomendação X.520. O diretório em si é estruturado na DIB na forma de árvore (denominada DIT: Directory Information Tree) conforme ilustrado na figura 8.7.

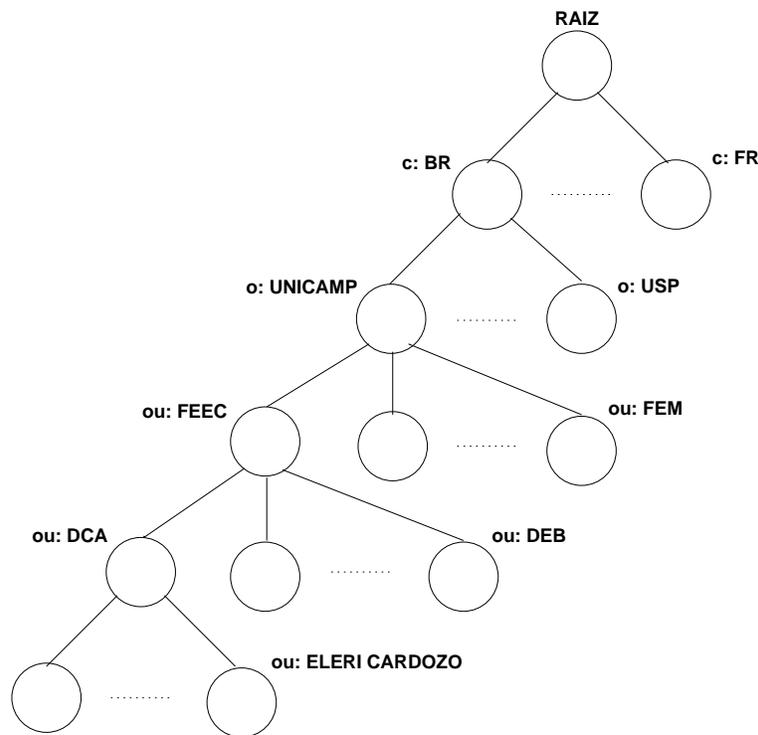


Figura 8.7: DIT (Directory Information Tree).

Do ponto de vista funcional, o DS é composto que quatro componentes. O primeiro, DSA (Directory Service Agent) armazena uma parte do diretório distribuído. O segundo componente, DUA (Directory User Agent) intermedia o acesso ao diretório por parte de um usuário. O terceiro componente, DAP (Directory Access Protocol) viabiliza a interação DUA-DSA; sendo o quarto componente, DSP (Directory Service Protocol) um protocolo de interação DSA-DSA. DAP e DSP são definidos como operações ROSE. Este modelo funcional é ilustrado na figura 8.8.

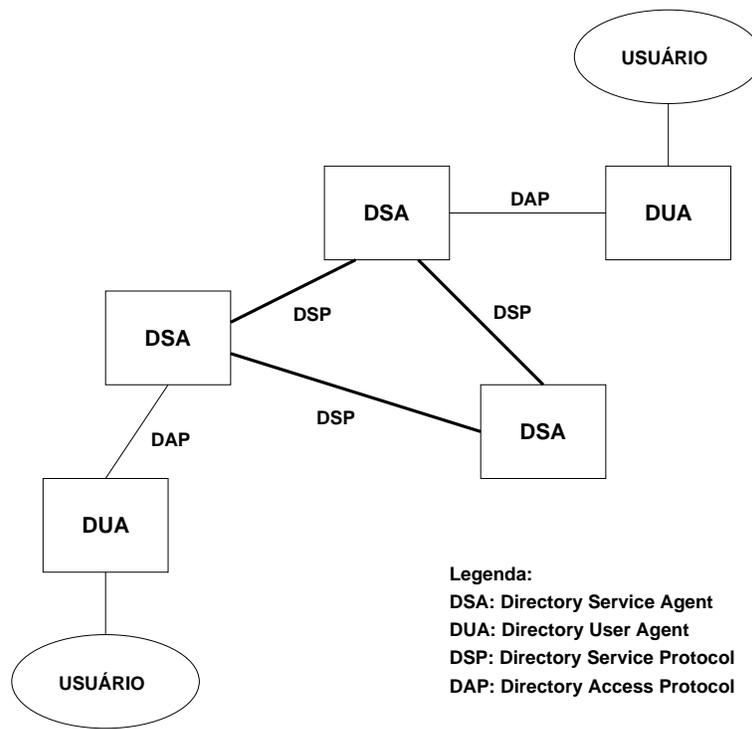


Figura 8.8: Modelo funcional do Serviço de Diretório (DS)

O DAP é um protocolo tipo requisição-resposta. A requisição carrega tipicamente um nome distinto de um objeto e a resposta o conteúdo deste objeto. Caso o objeto não esteja armazenado no DSA, este pode:

1. retornar o endereço de apresentação de outro DSA “mais próximo” da informação requisitada;
2. encadear a solicitação para outro DSA;
3. difundir a solicitação para um conjunto de DSAs, repassando uma eventual resposta ao DUA que solicitou a informação.

8.9 Serviço de Transferência de Arquivos (FTAM)

FTAM (File Transfer, Access and Management) é um ASE que provê funcionalidades para acesso, transferência e gerência de arquivos remotos, diretórios e *links* simbólicos⁹. O FTAM define um sistema de arquivos “virtual”, independente do sistema de arquivos “real” da máquina. As operações FTAM são definidas sobre o primeiro e mapeadas no segundo de acordo com a implementação (figura 8.9).

O FTAM utiliza o ACSE para o estabelecimento de uma sessão de operação sobre arquivos remotos e o CCR para agrupar em uma mesma sessão uma sequência de operações.

⁹Diretórios e links simbólicos foram incorporados como adendos à especificação em 1990.

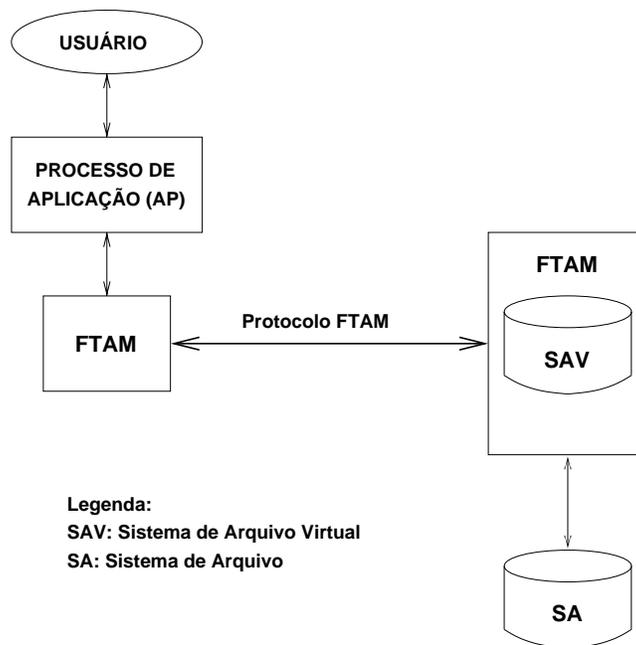


Figura 8.9: FTAM: Sistema de Arquivos Virtual.

Um arquivo é composto de zero ou mais unidades de dados (DU: Data Units). As DUs são organizadas em uma estrutura hierárquica (árvore) que determina a inter-relação entre as DUs. Esta estrutura é denominada FADU (File Access Data Unit). Caso o arquivo possua apenas uma FADU, o mesmo é denominado *não-estruturado*; se a hierarquia de FADUs possuir apenas 2 níveis, o arquivo é dito *sequencial*; com mais de 2 níveis, tem-se um arquivo *estruturado em blocos*. A figura 8.10 ilustra estas três possibilidades.

O FTAM define nove tipos de arquivos, desde arquivos de texto não estruturados até arquivos especiais para aplicações gráficas.

O FTAM possui 25 primitivas. Seis primitivas são definidas para abertura/fechamento de sessão:

1. F-INITIALIZE: inicia uma sessão FTAM (serviço confirmado);
2. F-TERMINATE: encerra uma sessão FTAM (serviço confirmado);
3. F-U-ABORT: interrompe (aborta) uma sessão FTAM (serviço não confirmado);
4. F-P-ABORT: o provedor de serviço interrompe (aborta) uma sessão FTAM (primitiva tipo indicação apenas);
5. F-SELECT: seleciona um arquivo para operação (serviço confirmado);
6. F-DESELECT: desfaz uma seleção (serviço confirmado).

Quatro primitivas são definidas para gerenciamento de arquivos (todas as primitivas são serviços confirmados):

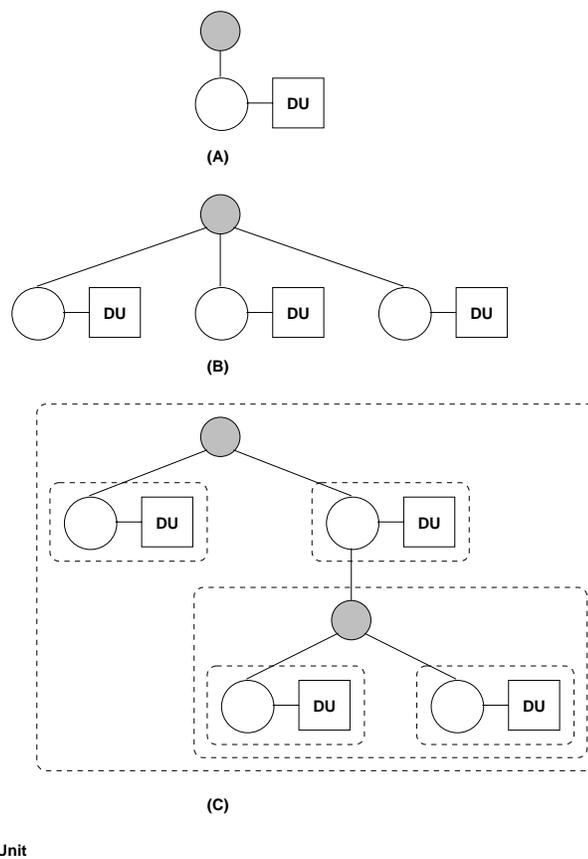


Figura 8.10: FTAM: Arquivos não estruturado (A); sequencial (B); estruturado em blocos (C). As linhas pontilhadas do item (C) definem FADUs.

1. F-CREATE: cria um novo arquivo;
2. F-DELETE: remove um arquivo.
3. F-READ-ATTRIB: lê os atributos do arquivo (permissões, tipo, etc.);
4. F-CHANGE-ATTRIB: altera os atributos do arquivo.

Seis primitivas são utilizadas para gerenciamento de falhas e atomicidade (todas as primitivas são serviços confirmados):

1. F-BEGIN-GROUP: marca o início de uma operação atômica;
2. F-END-GROUP: sinaliza o término de uma operação atômica;
3. F-RECOVER: restabelece um regime de transferência após a ocorrência de falha;
4. F-CANCEL: termina abruptamente uma transferência de arquivo;
5. F-CHECK: estabelece um *checkpoint* no regime de transferência¹⁰;

¹⁰O FTAM não utiliza pontos de sincronização para este serviço.

6. F-RESTART: retorna a um *checkpoint* anterior.

Finalmente, nove primitivas oferecem o serviço de transferência propriamente dito:

1. F-OPEN: abre um arquivo para leitura/escrita (serviço confirmado);
2. F-CLOSE: fecha um arquivo aberto (serviço confirmado);
3. F-LOCATE: posiciona o ponteiro de transferência em uma FADU específica (serviço confirmado);
4. F-ERASE: destrói uma FADU (serviço confirmado);
5. F-READ: lê uma ou mais FADUs (serviço sem confirmação);
6. F-WRITE: escreve em uma ou mais FADUs (serviço sem confirmação);
7. F-DATA: sinaliza à entidade respondedora a presença de dados (primitiva tipo indicação apenas);
8. F-DATA-END: sinaliza à entidade respondedora o término de transferência de uma FADU (primitiva tipo indicação apenas);
9. F-TRANSFER-END: encerra o regime de transferência (serviço confirmado).

A figura 8.11 ilustra um diagrama de estados típico de operação do FTAM.

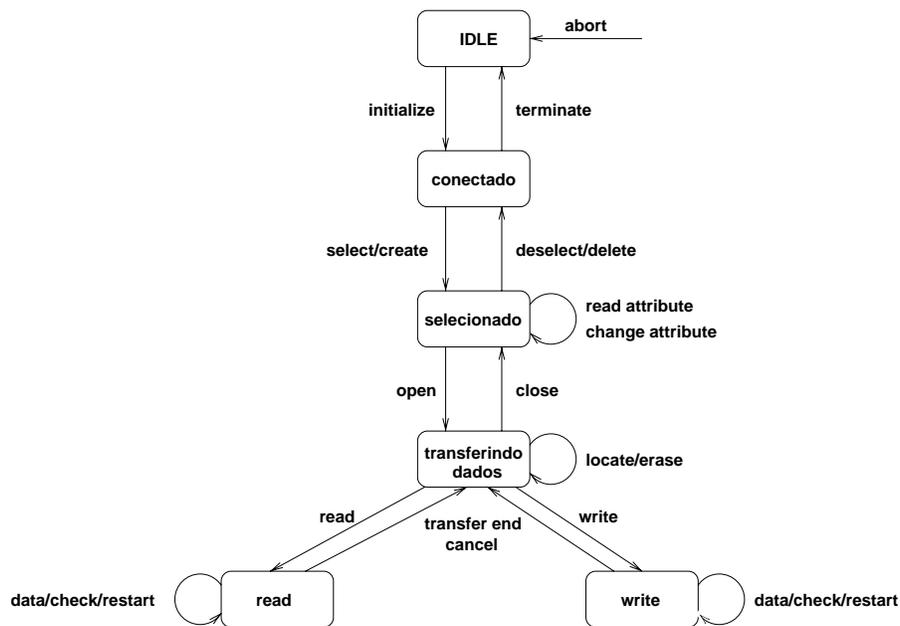


Figura 8.11: FTAM: diagrama de estados típico.

8.10 Terminal Virtual (VT)

VT (Virtual Terminal) é um ASE que define terminais virtuais utilizados para interação via rede com aplicações remotas. A necessidade de tal serviço está relacionada à vasta gama de terminais existentes no mercado. Ao invés de incorporar as particularidades de cada terminal, uma aplicação opera sobre este terminal virtual e uma outra entidade mapeia este dispositivo virtual em um dispositivo real. A idéia é similar ao conceito de sistema de arquivos virtual (figura 8.9) do FTAM.

A ISO define várias classes de terminais de acordo com a complexidade dos objetos que o terminal é capaz de manipular. Atualmente, apenas a classe referente aos terminais baseados em caractere está padronizada no âmbito do VT.

O VT é um ASE similar aos descritos anteriormente: utiliza o ACSE e uma conexão de apresentação para o estabelecimento de associação e transporte de PDUs, respectivamente. O VT utiliza também os serviços de sessão para organização do diálogo (via token) caso as entidades estabeleçam comunicação alternada nos dois sentidos (TWA).

O protocolo VT prevê:

1. estabelecimento de associação entre as entidades VT¹¹;
2. término normal ou abrupto da associação;
3. transferência de dados pela associação;
4. negociação e renegociação de um perfil de operação para o terminal virtual;
5. gerenciamento do diálogo¹².

As primitivas para abertura e encerramento de associação são idênticas às do ACSE com o prefixo VT: VT-ASSOCIATE, VT-RELEASE, VT-U-ABORT, VT-P-ABORT.

A primitiva principal para transferência de dados, VT-DATA, define um serviço sem confirmação. Um dos parâmetros desta primitiva é a prioridade dos dados: normal, alta ou urgente. Um par de primitivas também sem confirmação, VT-DELIVER e VT-ACK-RECEIPT são utilizadas, respectivamente, para transferência de dados e reconhecimento da recepção.

A negociação do perfil de operação se dá através das seguintes primitivas:

- VT-START-NEG: solicita à outra entidade o início de uma sessão de negociação (serviço confirmado);
- VT-OFFER-NEG: consulta à outra entidade sobre a conveniência de se estabelecer uma sessão de negociação (serviço sem confirmação);
- VT-ACCEPT-NEG/VT-REJECT-NEG: responde afirmativa ou negativamente à consulta acima (serviço sem confirmação);

¹¹Tipicamente uma vinculada ao terminal do usuário e outra vinculada à aplicação remota.

¹²O protocolo VT define dois modos de operação: no modo A inexistente qualquer forma de gerenciamento, enquanto no modo B o gerenciamento é baseado em *tokens*.

- VT-END-NEG: termina uma sessão de negociação (serviço confirmado);
- VT-SWITCH-PROFILE: altera o perfil de operação de um terminal virtual (serviço confirmado). Um ponto de sincronismo maior é inserido caso a associação seja reiniciada via VT-BREAK.

O gerenciamento do diálogo (modo B) se dá através das primitivas VT-REQUEST-TOKENS e VT-GIVE-TOKENS, ambas sem confirmação.

Finalmente, o protocolo VT oferece uma primitiva para reiniciar a conexão (via P-RESYNCHRONIZE): VT-BREAK. Este serviço é confirmado.

8.11 Problemas

1. Cite a filosofia de integração entre os ambientes local e OSI.
2. Como a camada de aplicação é estruturada?
3. Qual a diferença entre *conexão* e *associação*?
4. Qual a vantagem de se definir objetos de serviço (ASOs)?
5. Descreva a utilidade do ACSE, ROSE e RTSE.
6. Cite dois empregos típicos de transações atômicas.
7. Desenhe um diagrama temporal para uma transação atômica envolvendo coordenador e 3 participantes. Considere os casos de término normal e anormal (*rollback*).
8. Descreva funcionalmente o MHS.
9. Descreva funcionalmente o DS.
10. Descreva funcionalmente o FTAM.