Projeto de Protocolo para Rede de Alta Velocidade

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ELETRÔNICA

Autor:

_____

Ricardo Neimo Belem Lima

Orientador:

_____

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Examinador:

_____

Prof. Antonio Carneiro Mesquita Filho, Dr. Etat.

Examinador:

_____

Prof. Jorge Lopes de Souza Leão, Dr. Ing.

DEL

Março de 1996

AGRADECIMENTOS

Ao professor Aloysio de Castro Pinto Pedroza, pela atenção e incentivo dedicados à mim durante a fase em que estive na Iniciação Científica e para realização deste trabalho.

Aos professores Antonio Carneiro Mesquita Filho e Jorge Lopes de Souza Leão pela participação na banca examinadora.

Aos professores Mauros Campelo Queiroz e Gelson Vieira Mendonça por terem sido mais que professores, por terem sido amigos.

Aos meus amigos de turma pelo auxílio nas matérias e pelo convívio extra-classe.

Aos amigos Ildemar e Marlos por terem me socorrido em momentos difícieis na execução deste trabalho.

À minha família, pois sem seu apoio não teria chegado ao fim desta fase.

A todos, que direta ou indiretamente tenham me incentivado durante a conclusão deste curso.

RESUMO

Este trabalho tem como objetivo a especificação, geração de seqüências de teste e simulação de um protocolo de comunicações para redes de alta velocidade, tendo sido baseado na Recomendação I.363 da *International Tellecomunication Union (ITU)*, que consiste na padronização das funções e interações que um dado protocolo deve possuir para fazer uso de uma rede ATM. O protocolo foi especificado formalmente na linguagem Estelle e submetido a um processo de verificação em um ambiente CAD ESTELLE que incluiu a simulação com seqüências de testes geradas automaticamente.

SUMÁRIO

CAPÍTULO I - Introdução 9

I.1 Apresentação 9
I.2 Camada de Adaptação ATM (AAL) 11
I.3 Protocolo Escolhido 11
I.4 Técnicas de Projeto 12
I.5 Roteiro do Trabalho 13

CAPÍTULO II - Projeto do Protocolo AAL Tipo 3/4 14

II.1 Introdução 14
II.2 Camada AAL 14
II.2.1 Classes de Serviço 15
II.2.1.1 Classe A 15
II.2.1.2 Classe B 15
II.2.1.3 Classe C 16
II.2.1.4 Classe D 16
II.2.1.5 Classe X 16
II.2.2 Tipos de AAL 17
II.2.2.1 AAL 0 17
II.2.2.2 AAL 1 17
II.2.2.3 AAL 2 17
II.2.2.4 AAL 3/4 18
II.2.2.5 AAL 5 18

II.3 Descrição do AAL 18
II.4 Descrição do Serviço 20
II.4.1 Primitivas 21
II.4.1.1 Primitivas do AAL 21
II.4.1.1.1 AAL_Unitdata_request 21
II.4.1.1.2 AAL_Unitdata_indication 22
II.4.1.1.3 AAL_U_Abort_request 22
II.4.1.1.4 AAL_U_Abort_indication 22
II.4.1.1.5 AAL_P_Abort_indication 22
II.4.1.2 Primitivas do CPCS 22
II.4.1.2.1 CPCS_Unitdata_invoke 22
II.4.1.2.2 CPCS_Unitdata_signal 23
II.4.1.2.3 CPCS_U_Abort_invoke e CPCS_U_Abort_signal 23
II.4.1.2.4 CPCS_P_Abort_signal 23
II.4.1.3 Primitivas do SAR 24
II.4.1.3.1 SAR_Unitdata_invoke 24
II.4.1.3.2 SAR_Unitdata_signal 24
II.4.1.3.3 SAR_U_Abort_invoke e SAR_U_Abort_signal 25
II.4.1.3.4 SAR_P_Abort_signal 25
II.4.2 Funções, Estrutura e Codificação do AAL 26
II.4.2.1 Subcamada SAR 26
II.4.2.1.1 Funções da Subcamada SAR 26
II.4.2.1.2 Estrutura e Codificação de um SAR-PDU 27
II.4.2.2 Subcamada CPCS 28
II.4.2.2.1 Funções da Subcamada CPCS 29
II.4.2.2.2 Estrutura e Codificação de um CPCS-PDU 30
II.4.3 Procedimentos 32
II.4.3.1 Procedimentos da Subcamada SAR 32
II.4.3.1.1 Procedimentos da Subcamada SAR lado transmissor 32
II.4.3.1.2 Procedimentos da Subcamada SAR do lado receptor 34
II.4.3.2 Procedimentos da Subcamada CPCS 35
II.4.3.2.1 Procedimentos da Subcamada CPCS lado transmissor 36

II.4.3.2.2 Procedimentos da Subcamada CPCS do lado receptor

II.5 Especificação em Linguagem Estelle

II.5.1 Arquitetura da Especificação

II.5.2 Descrição da Especificação

II.6 Comentários

CAPÍTULO III - Resultados 44

III.1 Introdução 44

III.2 Geração Automática de Seqüência de Testes 44

III.3 Simulação 45

III.4 Comentários 46

CAPÍTULO IV - Conclusões 47

APÊNDICE A - Especificação formal em linguagem Estelle do protocolo AAL tipo 3/4

APÊNDICE B - Acompanhamento de uma sessão de testes

LISTA DE FIGURAS

II.1	Estrutura do AAL	20
II.2	Primitivas do AAL	25
II.3	Estrutura de um SAR-PDU	27
II.4	Estrutura de um CPCS-PDU	30
II.5	Exemplos de utilização do campo PAD	31
II.6	Máquina de estados de SAR transmissor	33
II.7	Máquina de estados de SAR receptor	34
II.8	Máquina de estados de CPCS transmissor	36
II.9	Máquina de estados de CPCS receptor	38
II.10	Arquitetura da especificação do protocolo AAL	41
II.11	Arquitetura da especificação com módulos e canais de interação	42
III.1	Máquina de estados do serviço AAL	45
III.2	Seqüências de testes do serviço AAL	45

LISTA DE TABELAS

II.1	Codificação do campo *Segment Type (ST)*	28
II.2	Valores permitidos para o campo *Lenght Indication (LI)*	29
II.3	Definições dos estados de SAR transmissor	32
II.4	Definições dos estados de SAR receptor	34
II.5	Definições dos estados de CPCS transmissor	35
II.6	Definições dos estados de CPCS receptor	37

CAPÍTULO I

INTRODUÇÃO

I.1 Apresentação

Hoje existem muitas redes de telecomunicação heterogêneas em natureza e serviço específico. Por exemplo, considere-se os casos de telefonia convencional, telex, TV's convencionais e a cabo, transmissão de dados em redes de computadores, e tantos outros.

Cada rede é introduzida para suportar uma bem definida e específica aplicação de modo a otimizar sua utilização. Porém, a tecnologia usada nestas redes não oferece uma solução comum de rede para abranger todo o espectro de aplicações. Este problema de integração aumenta cada vez mais, devido à necessidade crescente de aumento da capacidade de transmissão das aplicações existentes e da criação de novas aplicações, como vídeo conferência, biblioteca de vídeos, vídeo telefonia, e outras. Adotar uma solução particular de tecnologia de rede para cada tipo de aplicação tem muitas desvantagens, tais como ineficiência na utilização da capacidade da rede e inflexibilidade com futuros avanços tecnológicos.

Um primeiro passo para a rede de serviços integrados surgiu com a criação da **Rede Digital de Serviços Integrados de Banda Estreita (RDSI-BE)**, capaz de integrar voz, dados, e vídeo de baixa definição (64 kb/s até 2 Mb/s) num acesso de rede digital de voz. Entretanto, a RDSI-BE tem limitações, não podendo oferecer serviços que necessitem de altas velocidades de transmissão.

Uma rede universal, isto é, uma rede de serviços integrados deve ser capaz de suportar um largo número de serviços, tais como os que seguem:

- Velocidade baixa (telemetria, telecontrole, telealarme, voz, telefax, transmissão de dados de baixa velocidade);
- Velocidade média (*hi-fi sound*, vídeo telefonia);
- Transmissão de dados de alta velocidade;
- Velocidade muito alta (distribuição de vídeo de alta qualidade, biblioteca de vídeos, vídeos educacionais).

Portanto, o modo de transferência para esta rede universal não pode ser designado especificamente por um serviço. Verifica-se que existe uma larga faixa de serviços, com uma estimada taxa de transmissão de uns poucos bits/s até algumas centenas de Mb/s. Claramente, faz-se necessária uma rede de alta performance e alta velocidade. Este é o objetivo da **Rede Digital de Serviços Integrados de Banda Larga (RDSI-BL)**.

O **CCITT (*The International Consulative Commitee for Telecomunications and Telegraphy*)**, um orgão internacional de padronização técnica sediado na França, é o responsável pelas recomendações (padrões) da **Rede Digital de Serviços Integrados (RDSI)** e adotou o **Modo de Transferência Assíncrono (ATM)** como solução para a implementação de redes RDSI-BL devido às vantagens que esta tecnologia possui, tais como flexibilidade, independência da taxa de transmissão e comutação de todos os serviços em uma estrutura única.

A tecnologia ATM é capaz de suportar diferentes serviços para satisfazer aos requisitos exigidos pelos diferentes tipos de tráfego, a altas velocidades de transmissão. Redes baseadas em ATM podem ser projetadas de forma a aproveitar melhor os meios de comunicação na presença de tráfegos em rajadas e, ao mesmo tempo, garantir retardo máximo para serviços que suportam fontes de tráfego contínuo.

Por isso o Modo de Transferência Assíncrono foi a tecnologia escolhida para suportar a diversidade de serviços definida para a RDSI-BL.

De acordo com o CCITT, as camadas do modelo ATM são três: a camada **PHY (*Phisical* - Física)**, que primordialmente transporta informação; a camada **ATM**, que tem como funções principais chaveamento/roteamento e multiplexação; e a **camada de adaptação ATM (AAL)**, cuja responsabilidade principal é adaptar informações de serviço para o ATM. A camada física e a camada ATM não serão descritas resumidamente, pois a camada AAL é que será alvo deste trabalho e será vista detalhadamente adiante.

I.2 Camada de Adaptação ATM (AAL)

A Camada de Adaptação ATM tem como função efetuar as adaptações necessárias para compatibilizar e oferecer os serviços desejados pelas camadas superiores, utilizando tecnologia ATM.

Existem várias classes de serviços suportados por esta tecnologia e cada uma é provida de uma camada de adaptação diferente. Isto quer dizer que o seu propósito é acrescentar funcionalidades para cada camada de adaptação ATM, de modo a diferenciar as classes de serviços exigidas pelas camadas superiores.

Para que estas aplicações fossem padronizadas, o CCITT definiu grupos de serviços oferecidos pela AAL classificando-os segundo as seguintes características:

- Taxa de transmissão variável (VBR) ou constante (CBR);
- Orientado ou não à conexão;
- Necessidade ou não de se manter a relação temporal entre a fonte e o destino (serviços isócronos).

Destas considerações surgiram cinco classes, definidas pela recomendação I.362. A partir das classificações dos serviços, o ITU-T definiu os tipos de AAL para suporte a estas classes, através da recomendação I.363.

Posteriormente serão apresentadas as classes de serviços (classes A, B, C, D e X) e as camadas de adaptação (AAL 0, 1, 2, 3/4 e 5) em maiores detalhes.

I.3 Protocolo Escolhido

A importância da RDSI-BL e do ATM quando da integração de serviços existentes e também no caso do surgimento de novas aplicações é relevante e já foi mencionada. Disto resulta a importância das Camadas de Adaptação ATM (AAL) e também explica o desejo de se projetar um protocolo AAL que torne possível as comunicações das redes já existentes com as redes ATM.

O protocolo escolhido para especificação e implementação tem como base o AAL 3/4. Sua importância se deve à prestação de serviços orientados à conexão e também serviços não orientados à conexão.

Poderia ser escolhido para o projeto o AAL 5, um protocolo mais simples e poderoso, mas devido justamente ao fato de sua simplicidade, sua implementação não colocaria em teste todo o potencial de uma especificação mais abrangente.

Quanto ao AAL 3/4, ele possui uma certa dose de complexidade e isso o tornará ainda mais atrativo se o analisarmos sob o ponto de vista de uma especificação em Estelle mais completa.

I.4 Técnicas de Projeto

O comportamento de um protocolo como um todo é especificado em termos de Máquinas de Estado Finitas (MEF) para melhor entendimento. As MEF's simplesmente descrevem as ações que um protocolo deve tomar mediante o acontecimento de determinadas situações.

Este é o primeiro passo dessa especificação. Após se estudar suas funções, primitivas e comportamento, o AAL 3/4 é descrito em termos de MEF's.

O próprio uso de MEF's na descrição de protocolos é base para sua síntese, testes de validação e testes de conformância, três partes importantes no projeto de protocolos.

Neste trabalho foi usada a linguagem Estelle para se especificar formalmente o protocolo. Esta linguagem foi desenvolvida pela *International Standards Organization* (ISO), e é uma técnica de descrição formal baseada em um modelo de transição de máquinas de estados.

Quanto à parte de avaliação do desempenho, conta-se com o uso do Gerador Automático de Seqüências de Testes para Protocolos de Comunicação (GASTPC) [2], que nos fornece de modo automático uma seqüência de testes, a qual devemos submetê-la por meio do Simulador [4] para testar a total confiabilidade do protocolo em executar as funções propostas no início de sua implementação.

I.5 Roteiro do Trabalho

O Capítulo II apresenta a descrição completa do protocolo escolhido, o AAL 3/4, suas funções, primitivas, serviços, codificações de mensagens e máquinas de estado de suas subcamadas, bem como seu comportamento global. Apresenta também sua especificação em linguagem Estelle e todos os comentários a cerca desse passo. Mostra inclusive como o protocolo será testado posteriormente através de suas seqüências de testes e simulação.

O Capítulo III apresenta todos os resultados obtidos quando do teste funcional do protocolo durante sua implementação. Apresenta a geração de seqüências de testes e toda a parte de simulação, mostrando comentários a respeito de seu desempenho em relação ao especificado.

Finalmente no Capítulo IV são apresentadas algumas observações conclusivas e questões para futuros trabalhos.

CAPÍTULO II
Projeto do Protocolo AAL Tipo 3/4

II.1 Introdução

O protocolo AAL 3/4 provê transferência confiável de dados entre dois usuários que estejam utilizando como meio uma rede ATM. Inicialmente será feita uma descrição das classes de serviços suportados por esta tecnologia e das respectivas camadas de adaptação AAL existentes para proverem tais serviços. Posteriormente será feita uma abordagem detalhada do protocolo escolhido, que conterá a descrição das primitivas de serviço, bem como das PDU's utilizadas e também das respectivas máquinas de estados das subcamadas que compõem o serviço proposto por este protocolo. Ainda neste capítulo será dada uma breve apresentação da linguagem Estelle, seguida da especificação realizada. Por fim serão feitos comentários a respeito da geração de seqüências de testes e da simulação deste protocolo.

II.2 Camada AAL

Como mencionado no capítulo anterior, o ITU-T definiu classes de serviços suportados pelo ATM e dessas classes definiu também os tipos de AAL para dar suporte a elas. A seguir serão apresentadas as classes de serviços e as camadas de adaptação em maiores detalhes.

II.2.1 Classes de Serviço

II.2.1.1 Classe A

É utilizada para emulação de circuitos, ou seja, aplicações onde existe uma relação temporal entre fonte e destino (serviços isócronos), taxa de transmissão constante e o serviço é orientado à conexão.

São funções desta classe:

- Segmentação e remontagem da informação;
- Detecção e tratamento de perda, discarte, duplicação ou troca na seqüência de células ATM;
- Recuperação do *clock* (sincronização) da fonte (serviço isócrono);
- Detecção de erros nos bits no campo da informação do usuário;
- Não realiza compressão ou compactação.

Exemplos destes serviços são a transmissão de voz e vídeo a taxas constantes.

II.2.1.2 Classe B

É praticamente igual à classe A, com reproduções feitas à taxa constante, mas podendo ser codificadas com taxas variáveis através de compressão e compactação. Existe ainda a indicação de perda ou corrupção da informação, mas sem a preocupação de recuperação da mesma.

Exemplos destes serviços são tráfego de voz e vídeo codificados com taxas variáveis através de compressão e compactação.

II.2.1.3 Classe C

São serviços não isócronos (sem relação temporal entre fonte e destino) orientados à conexão, onde a variação estatística do retardo não causa maiores problemas. Também existem outros serviços tais como multiplexação e demultiplexação de vários usuários dentro de uma única conexão ATM.

As funções desta classe são:

- Segmentação e remontagem da informação;
- Detecção e sinalização de erros nos dados.

Um exemplo tradicional desta classe são os serviços em redes de comutação de pacotes com conexão, como **X.25**.

II.2.1.4 Classe D

São serviços sem conexão, não isócronos e com taxa variável.

Correspondem aos serviços sem conexão das redes de dados, como por exemplo os de interconexão de redes com TCP/IP.

Como os outros tipos de classes de serviço, esta classe provê:

- Segmentação e remontagem da informação;
- Detecção de erros nos dados (mas sem retransmissão).

II.2.1.5 Classe X

A classe X define um serviço orientado à conexão ATM, onde o tipo de tráfego, a taxa de transmissão (constante ou variável) e a sincronização são definidos pelo usuário.

II.2.2 Tipos de AAL

II.2.2.1 AAL 0

Este tipo não tem função associada, atende aos requisitos da classe X e liga o usuário diretamente a camada ATM. A AAL nula é usada quando objetiva-se deixar a cargo do equipamento fornecer os seus serviços utilizando diretamente a tecnologia de transferência baseada no ATM. Estes serviços passam então a ser responsabilidade das camadas superiores.

II.2.2.2 AAL 1

Atende aos serviços de classe A, ou seja, para tráfego de taxa constante (CBR), isócronos e orientado à conexão.

Suas funções são:

- Recebe a informação da origem e a divide em uma unidade do tamanho do campo que porta a informação dentro da célula, para que seja transmitida através da rede;
- Restaura a sincronização da origem ao entregar a informação ao usuário destino.

Esta segunda função é necessária, pois, quando a informação flui através da rede, ela não mais mantém um padrão contínuo devido à variação estatística do retardo, que pode fazer uma célula atrasar um determinado tempo, enquanto que a seguinte a ela atrasaria o dobro do tempo.

II.2.2.3 AAL 2

É utilizada para prover serviços do tipo classe B. Isto é, serviços para dados com tráfego com taxa variável. Atualmente ainda não existem recomendações ITU-T sobre a sua operação.

II.2.2.4 AAL 3/4

O AAL 3/4 presta serviços do tipo C (orientado à conexão) e D (não orientado à conexão). No início duas camadas foram propostas, a Camada de Adaptação ATM 3 (AAL 3) e a Camada de Adaptação ATM 4 (AAL 4), fornecendo suporte às classes C e D, respactivamente. Essas camadas se fundiram durante o processo de definição das normas, formando uma única camada AAL 3/4 devido à semelhança de vários procedimentos realizados por cada uma delas a fim de prestar os serviços designados.

II.2.2.5 AAL 5

O propósito de elaboração do AAL 5 foi o de reduzir o processamento de protocolo e o *overhead* de transmissão, diminuindo significativamente a complexidade de implementação. À exceção da multiplexação, todas as outras funções realizadas pelo AAL 3/4 podem também ser realizadas por esta camada.

Pode ser usado em serviços de dados orientados à conexão ou não. A transferência de dados é feita em operação assegurada e não assegurada, modo de mensagem ou fluxo, suportando deteção de erros, sem fazer recuperação de pacores perdidos.

II.3 Descrição do AAL

O protocolo que é mostrado neste trabalho foi baseado rigorosamente no AAL tipo 3/4 da Recomendação I.363 da ITU-T. Apenas foram feitas algumas considerações de modo a simplificar sua implementação, tais como, não utilização da correção de erros nas células entregues e não utilização da função de multiplexação.

O AAL 3/4 é formado por duas subcamadas: **Subcamada de Convergência (CS - *Convergence Sublayer*)** e **Subcamada de Quebra e Remontagem (SAR - *Segmentation and Reassembly Sublayer*)**.

As funções da subcamada CS são:

- Multiplexação;
- Detecção de perdas de células;
- Recuperação do relógio (relação temporal da origem para o destino).

Destas funções descritas, somente utilizaremos a que permite a detecção de células perdidas, isso para facilitar a implementação.

As funções da subcamada SAR são:

- Fragmentação da informação de modo a poder se acomodar no campo de informação das células ATM;
- Remontagem da informação fragmentada a partir das células recebidas.

A subcamada de convergência CS é subdividida em **Subcamada de Convergência Específica do Serviço (SSCS - *Service Specific Convergence Sublayer*)** e **Parte Comum da Subcamada de Convergência (CPCS - *Commom Part Convergence Sublayer*)**. A figura II.1 mostra melhor o que foi descrito.

A SSCS é a subcamada responsável pelos procedimentos específicos para cada classe de serviço e para serviços específicos de uma mesma classe. Fornece funções adicionais conforme a necessidade de serviços específicos. Conforme a Recomendação I.363 da ITU-T, ainda não foram descritos procedimentos para implementação da SSCS, portanto esta subcamada será considerada nula.

A CPCS efetua os serviços comuns às classes de serviço. Adiciona cabeçalhos aos quadros do usuário e assegura integridade no nível de quadros.

As interfaces entre as subcamadas do AAL não possuem pontos de acessos a serviços (SAP - *Service Acess Point*) pois são definidas apenas através de primitivas lógicas e, com isso, não estão disponíveis ao usuário.

As unidades de informação que fluem entre as camadas tem a seguinte terminologia:

- Unidade de Informação de Serviço (SDU - *Service Data Unit*): é recebida por uma camada através da interface com sua camada superior;
- Informação de Controle de Protocolo (PCI - *Protocol Control Information*);

- Unidade de Informação de Protocolo (PDU - *Protocol Data Unit*): é entregue à camada inferior e é composta por uma SDU recebida da camada superior, adicionada à sua respectiva PCI.

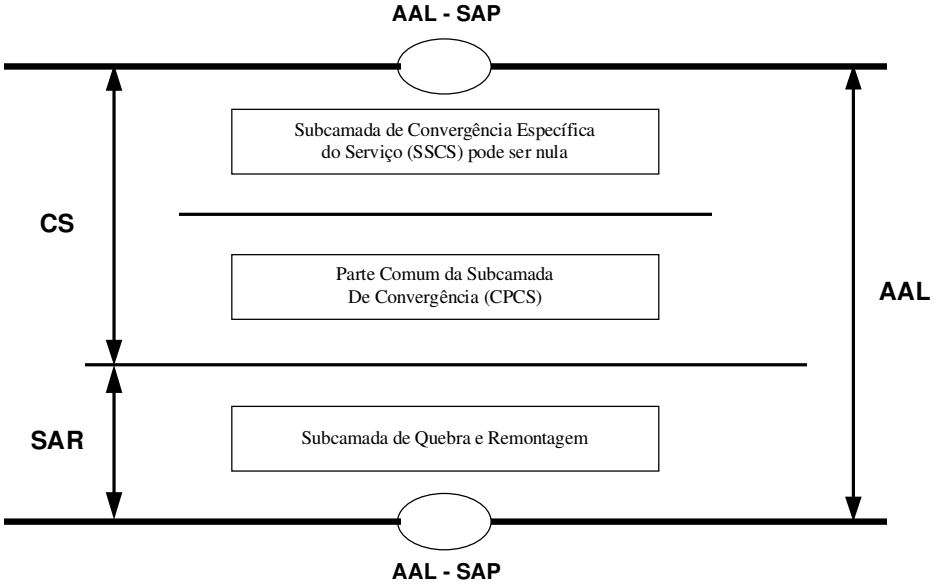


Figura II.1- Estrutura do AAL

II.4 Descrição do Serviço

O AAL 3/4 oferece dois modos de serviços de transporte (transferência de dados):

- **Modo de Mensagem (*Message Mode Service*)**: um quadro de informação é recebido do usuário e transportado a outro usuário pela rede;
- **Modo de Fluxo (*Streaming Mode Service*)**: o fluxo de quadros é transportado de um usuário a outro pela rede.

Ao transportar células através de uma rede ATM, algumas poderão ser descartadas, perdidas, entregues erradamente ou danificadas. Dependendo da forma escolhida para ser feito o tratamento das células recebidas, decisões diferentes serão tomadas em cada uma das situações.

- Entregar somente SDU's corretas à camada superior, descartando qualquer SDU que teve células perdidas ou danificadas;
- Fazer correção de erros e entregar todas as SDU's completas;
- Não fazer correção, mesmo assim entregar todas as SDU's completas. Neste caso, erros detectados são sinalizados.

Para efetuar as decisões descritas acima existem dois tipos definidos de operações. Na **Operação Assegurada** é feita a recuperação de células com erros através de retransmissão. Na **Operação Não Assegurada**, o AAL não efetua a recuperação de erros. O usuário decide se recebe ou não SDU's com erro.

Na implementação deste protocolo, optou-se por escolher o modo de serviço *message mode*, pois na Recomendação I.363, ainda não havia regulamentação para os procedimentos adotados pelo CPCS quando da utilização do *streaming mode*. Optou-se também pela *operação não assegurada* aonde não existe recuperação de erros. Nesta implementação, quando há uma perda de célula ou célula trocada, imediatamente ocorre uma sinalização ao usuário e uma primitiva *Abort* é enviada, cessando a transmissão do quadro.

II.4.1 Primitivas

II.4.1.1 Primitivas do AAL

II.4.1.1.1 AAL_Unitdata_request

Parâmetro: dados do usuário.

Esta primitiva é enviada ao SAP externo pela camada imediatamente superior ao AAL, e contém os dados que o usuário AAL transmissor (*sender*) deseja enviar ao usuário AAL receptor (*receiver*).

II.4.1.1.2 AAL_Unitdata_indication

Parâmetro: dados do usuário.

Esta primitiva é recebida pelo SAP externo vinda da subcamada CPCS e contém os dados enviados pelo usuário AAL transmissor.

II.4.1.1.3 AAL_U_Abort_request

Esta primitiva não é usada no message mode.

II.4.1.1.4 AAL_U_Abort_indication

Esta primitiva não é usada no message mode.

II.4.1.1.5 AAL_P_Abort_indication

Esta primitiva não é usada no message mode.

II.4.1.2 Primitivas do CPCS

Como não existem SAP's entre as camadas do AAL, as primitivas são designadas por *"invoke"* e *"signal"* ao invés do convencional *"request"* e *"indication"* justamente para realçar a ausência dos SAP's.

II.4.1.2.1 CPCS_Unitdata_invoke

Parâmetros: → *Interface Data (ID)*: corresponde aos dados recebidos pelo SAP externo;

→ *More (M)*: esse parâmetro não é usado no message mode;

→ *Maximum Lenght (ML)*: esse parâmetro não é usado no message mode;

→ *Reception Satus (RS)*: esse parâmetro não será usado, pois ele indica se o dado contém algum erro.

Esta primitiva é recebida pelo SAP do CPCS vindo do SAP externo do AAL. Esta primitiva nada mais é do que a primitiva AAL_Unitdata_request só que com o nome trocado.

II.4.1.2.2 CPCS_Unitdata_signal

Parâmetros: os mesmos do CPCS_Unitdata_invoke.

Esta primitiva é enviada pelo SAP do CPCS para o SAP externo do AAL. Esta primitiva nada mais é do que a primitiva AAL_Unitdata_indication só que com o nome trocado.

II.4.1.2.3 CPCS_U_Abort_invoke e CPCS_U_Abort_signal

Estas primitivas não são usadas no message mode.

II.4.1.2.4 CPCS_P_Abort_signal

Esta primitiva não é usada no message mode.

II.4.1.3 Primitivas do SAR

Estas primitivas modelam a troca de informações entre a subcamada SAR e a subcamada CPCS.

Como não existem SAP's entre as camadas do AAL, as primitivas são designadas por *"invoke"* e *"signal"* ao invés do convencional *"request"* e *"indication"* justamente para realçar a ausência dos SAP's.

II.4.1.3.1 SAR_Unitdata_invoke

Parâmetros: → *Interface Data (ID)*: corresponde aos dados trocados entre as subcamadas SAR e CPCS;

→ *More (M)*: esse parâmetro especifica se é o fim do SAR-PDU;

→ *Reception Satus (RS)*: esse parâmetro não será usado. Ele indica se o dado contém algum erro.

Esta primitiva transporta os dados da subcamada CPCS para a subcamada SAR onde haverá a execução da segmentação destes dados, para possibilitar o transporte dos mesmos através das células ATM.

II.4.1.3.2 SAR_Unitdata_signal

Parâmetros: os mesmos do SAR_Unitdata_invoke.

Esta primitiva é enviada do SAR para o CPCS transportando os dados do usuário.

II.4.1.3.3 SAR_U_Abort_invoke e SAR_U_Abort_signal

Parâmetro: nenhum.

Estas primitivas são usadas pelo usuário SAR para requerer o serviço de *Abort*. Também é usada pela entidade SAR para sinalizar ao usuário SAR que houve perda ou troca de células contendo informações.

II.4.1.3.4 SAR_P_Abort_signal

Parâmetro: nenhum.

Esta primitiva é usada pela entidade SAR para sinalizar o usuário que um SDU parcialmente entregue está sendo descartado devido a presença de algum erro.

A Figura II.2 mostra uma melhor visualização das primitivas e das subcamadas.

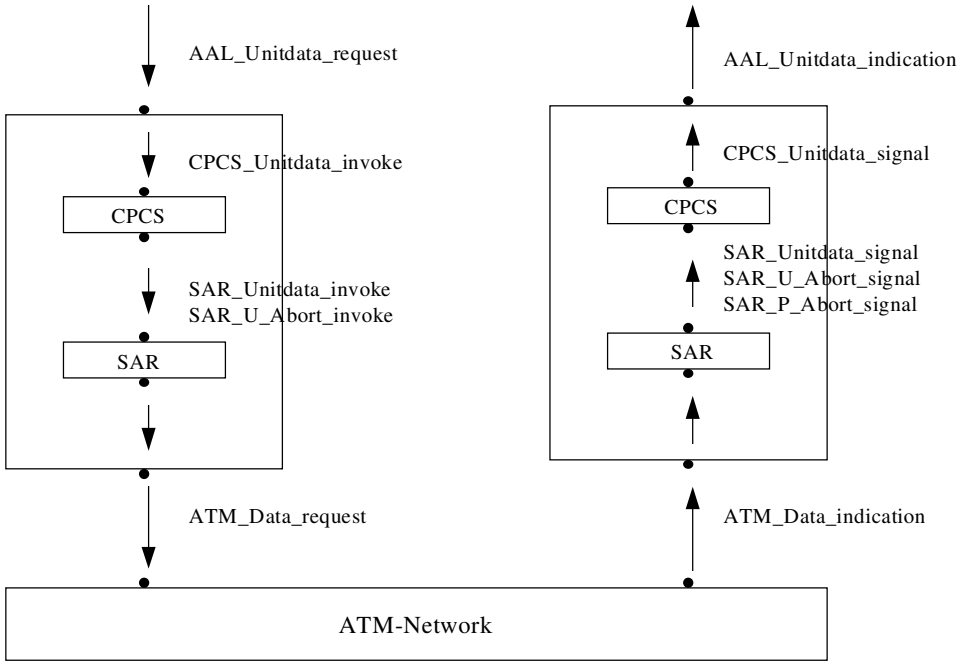


Figura II.2 - Primitivas do AAL

II.4.2 Funções,Estrutura e Codificação do AAL

II.4.2.1 Subcamada SAR

II.4.2.1.1 Funções da subcamada SAR

As funções da subcamada SAR são realizadas sobre um SAR-PDU básico. O SAR aceita tamanhos variáveis de SAR-SDU's oriundos do CPCS e gera SAR-PDU's contendo 44 octetos de dados e mais 4 octetos para cabeçalho (*header*) e cauda (*trailer*) (isso será mostrado no próximo ítem).

Essas funções provêm os meios necessários para transferência de múltiplos SAR-SDU's de tamanho variáveis através de uma conecção ATM entre entidades AAL.

1. **Preservação do SAR-SDU**

Essa função preserva o SAR-SDU através dos campos *Segment Type (ST)* e *Lenght Indication (LI)*. O *LI* indentifica o número de octetos do SAR-SDU contidos dentro do SAR-PDU. O *ST* identifica um SAR-PDU como sendo o começo de uma mensagem *(BOM)*, continuação de uma mensagem *(COM)*, fim de uma mensagem *(EOM)* ou como uma mensagem única *(SSM)*. Todos esses campos e essas siglas serão descritas com mais detalhes no próximo ítem.

2. **Detecção de Erro e Tratamento**

Essa função provê os meios necessários para detecção e tratamento de erros nos bits no próprio SAR-PDU e perda ou ganho de SAR-PDU's.

3. **Integridade da Seqüência do SAR-SDU**

Essa função assegura que a seqüência do SAR-SDU será mantida dentro da conexão.

4. **Multiplexação e Demultiplexação**

Essa função não será usada. Permite a inclusão de vários usuários dentro de uma única conexão ATM.

5. ***Abort***

Essa função provê os meios necessários para se abortar um SAR-SDU parcialmente transmitido,seja por perda, troca ou descarte de células ATM ou por qualquer detecção de erro no cabeçalho ou na cauda do SAR-PDU.

II.4.2.1.2 Estrutura e codificação de um SAR-PDU

Para implementação das funções da subcamada SAR é necessário a existência de um cabeçalho de 2 octetos e uma cauda também com 2 octetos. O cabeçalho e a cauda juntos com os 44 octetos de dados do SAR-PDU formam os 48 octetos que compõem a célula ATM (ATM-PDU).

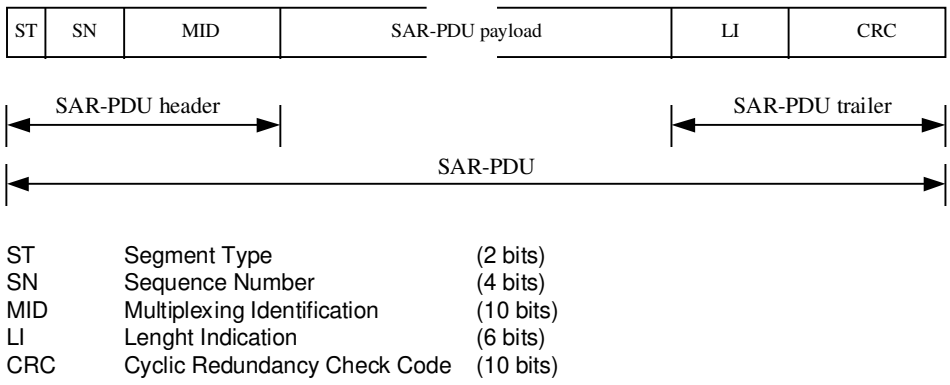


Figura II.3 - Estrutura de um SAR-PDU

a) **Campo *Segment Type (ST)***

Esse campo possui 2 bits e identifica um SAR-PDU como contendo um começo de mensagem *(BOM - beginning of message)*, uma continuação de mensagem *(COM - continuation of message)*, como fim de uma mensagem *(EOM - end of message)* ou como

segmento único de mensagem *(SSM - single segment)*. A associação entre sua codificação e seu tipo é mostrado na tabela abaixo.

Segment Type	Codificação	Tipo
BOM	10	Começo de mensagem
COM	00	Continuação da mensagem
EOM	01	Fim da mensagem
SSM	11	Segmento único

Tabela II.1 - Codificação do campo *Segment Type (ST)*

b) Campo *Sequence Number (SN)*

Esse campo permite identificar a seqüência correta de transmissão das células ATM. Cada SAR-PDU pertencente a um SAR-SDU terá seu *SN* incrementado de um em relação a um valor inicialmente escolhido. O receptor verifica a seqüência do campo *SN* dos SAR-PDU's derivados de um SAR-SDU. A cada novo SAR-SDU, o transmissor deverá ajustar o valor inicial do campo *SN*. Este campo possui 4 bits, ou seja seu valor varia de 0 a 15.

c) Campo *Multiplexing Identification (MID)*

Esse campo não será usado nesta implementação, pois não se está usando a função de multiplexação. Seu valor será fixado em zero.

d) Campo de dados *(SAR-PDU payload)*

Esse campo possui a informação propriamente dita, são os dados que o usuário deseja transportar ao outro lado da conexão.

e) Campo *Lenght Indication (LI)*

Esse campo é codificado de acordo com o número de octetos da informação do SAR-SDU que são incluidos no campo de dados do SAR-PDU (payload). Valores permitidos desse campo, de acordo com a codificação do campo *ST*, são mostrados na tabela abaixo.

Segment Type (ST)	Valores Permitidos para LI
BOM	44
COM	44
EOM	4 a 44
SSM	8 a 44

Tabela II.2 - Valores permitidos para o campo *Lenght Indication (LI)*

f) Campo *CRC*

Esse campo possui uma seqüência de 10 bits, usados para detectar erros em bits dentro do SAR-PDU. Como não se está utilizando esta função, esse campo não será usado e seu valor será fixado em zero.

II.4.2.2 Subcamada CPCS

II.4.2.2.1 Funções da subcamada CPCS

As funções da subcamada CPCS são realizadas sobre um CPCS-PDU. Conforme mencionado antes, tanto o CPCS como todo AAL foi idealizado para trabalhar no *message mode*. As funções implementadas pelo CPCS incluem:

1. Preservação do CPCS-SDU

Essa função provê meios para que um SDU percorra todo caminho até seu destino sem que haja violação de seu conteúdo.

2. Detecção de Erro e Tratamento

Quando da detecção de erros em algum CPCS-SDU, o usuário é sinalizado e a informação ignorada. Exemplos de erros podem ser: desigualdade entre *Btag/Etag*, transbordo (*overflow*) do *buffer* de remontagem, formato impróprio do CPCS-PDU ou erros indicados pela subcamada SAR.

3. **Alocação do *Buffer***

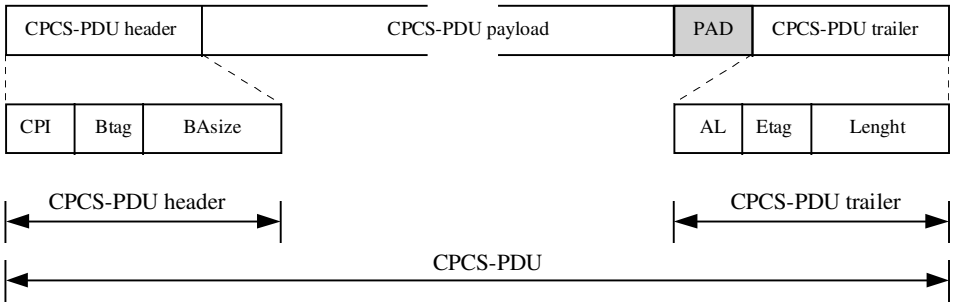
Essa função provê os meios necessários para alocação de espaço condizente com o CPCS-PDU que se está recebendo.

4. ***Abort***

Essa função provê meios necessários para se abortar um CPCS-SDU parcialmente transmitido, seja por perda, troca ou descarte de um CPCS-PDU ou por qualquer detecção de erro no cabeçalho ou na cauda do mesmo.

II.4.2.2.2 Estrutura e Codificação de um CPCS-PDU

Para implementação das funções da subcamada CPCS é necessário a existência de um cabeçalho de 4 octetos e de uma cauda também de 4 octetos.



CPI	Common Part Indicator	(1 octeto)
Btag	Beginning Tag	(1 octeto)
BAsize	Buffer Allocation Size	(2 octetos)
PAD	Padding	(0..3 octetos)
AL	Alignment	(1 octeto)
Etag	End Tag	(1 octeto)
Lenght	Lenght do CPCS-PDU payload	(2 octetos)

Figura II.4 - Estrutura de um CPCS-PDU

a) **Campo *Commom Part Indicator (CPI)***

Esse campo seria usado para interpretar campos subsequentes das funções CPCS no cabeçalho ou na cauda. Poderia ser uma unidade de contagem para outros campos, mas não será utilizado, será fixado como zero.

b) **Campo *Beginning Tag (Btag)***

Esse campo permite a associação entre o cabeçalho e a cauda do CPCS. O CPCS transmissor insere o mesmo valor para *Btag* e para *Etag* na cauda, e muda o valor para cada CPCS-PDU. O CPCS receptor verifica o valor de *Btag* no cabeçalho com o valor de *Etag* na cauda.

c) **Campo *Buffer Allocation Size Indication (BAsize)***

O campo *BAsize* indica ao receptor o espaço máximo necessário para alocação de um *buffer* capaz de receber o CPCS-SDU transmitido. No caso deste trabalho (*message mode*), o valor de *BAsize* é igual ao tamanho de dados do CPCS-PDU (payload).

d) **Campo *Padding (PAD)***

Esse campo simplesmente complementa o CPCS-PDU payload para que ele se torne um múltiplo de 4 octetos. Ele é ajustado com valor "0", não carregando nenhuma informação e será ignorado pelo receptor quando da remontagem da informação.

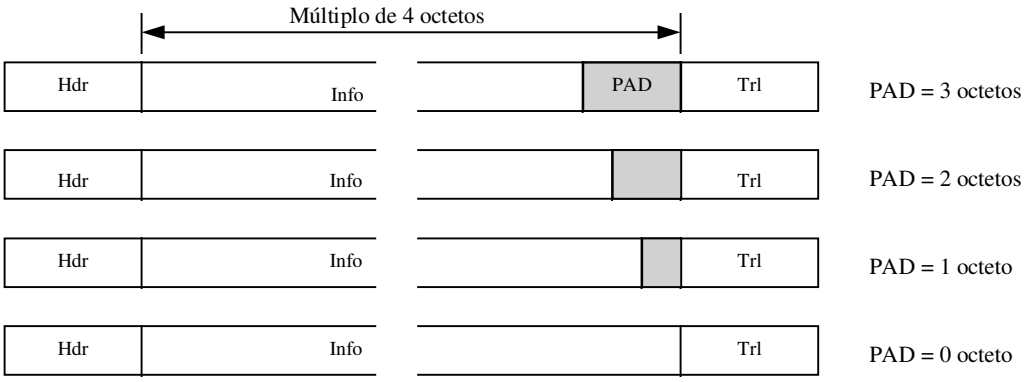


Figura II.5 - Exemplos de utilização do campo PAD

e) **Campo *Alignment (AL)***

Esse campo será ajustado com o valor zero.

f) Campo *End Tag (Etag)*

O CPCS transmissor insere o mesmo valor nesse campo como foi inserido no *Btag* do cabeçalho. Isso permitirá uma verificação posterior.

g) Campo *Lenght*

Esse campo é usado para codificar o tamanho do CPCS-PDU payload. É também utilizado para detectar perda ou ganho de informações.

II.4.3 Procedimentos

II.4.3.1 Procedimentos da subcamada SAR

II.4.3.1.1 Procedimentos da subcamada SAR do lado transmissor

Será mostrado neste ítem todo comportamento do protocolo referente à subcamada SAR, quando se comportando como transmissor. Partindo de sua máquina de estados, será observado quais decisões devem ser tomadas pelo protocolo, quando do recebimento de determinadas primitivas. Abaixo estão, sua máquina de estados e uma tabela discriminando seus estados.

Estado	Definição
IDLE	Esperando para transmitir nova SAR-SDU

Tabela II.3 - Definições dos estados de SAR transmissor

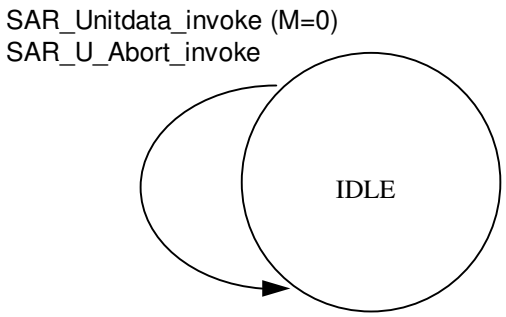


Figura II.6 - Máquina de estados de SAR transmissor

A função principal da subcamada SAR transmissor é a de segmentar o SAR-SDU oriundo da subcamada CPCS. Esta segmentação é necessária pois o ATM trabalha com transmissão de células de tamanho fixo contendo 48 octetos, logo o principal objetivo desta subcamada é de efetuar a compatibilização da mensagem recebida com o meio a ser utilizado para seu transporte. Isso é efetuado a partir da divisão do SAR-SDU em "pedaços" de 44 octetos e a inclusão de um cabeçalho (2 octetos) e uma cauda (2 octetos) para assegurar uma transmissão segura e sem falhas.

O SAR transmissor faz uso de uma variável de estado auxiliar chamada *snd_SN*, que tem a função de ajustar o campo *sequence number (SN)* localizado no cabeçalho do SAR-PDU. Esta variável é incrementada sempre que houver a remessa de um SAR-PDU para a camada ATM para transmissão. Quando do final da transmissão de um SAR-SDU, esta variável é reajustada.

Quando do estabelecimento da conexão SAR, o SAR transmissor procederá sempre para o estado IDLE (estado inicial). Sempre que se dirigir novamente a este estado, deve ser efetuada a mudança do valor da variável auxiliar *snd_SN* para um valor entre 0 e 15, isto significa que um SAR-SDU foi completamente transmitido e agora será efetuada a transmissão de um novo SAR-SDU.

Para cada SAR-PDU, o campo *SN* recebe o valor da variável *snd_SN* e depois dessa operação, esta variável deve ser incrementada de um (módulo 16).

Quando do recebimento da primitiva *SAR_Unitdata_invoke* do CPCS, o SAR transmissor começará o processo de segmentação. Se o *Interface Data (ID)* (parâmetro que carrega a informação) tiver um tamanho maior que 44 octetos, o SAR transmissor gerará mais de um SAR-PDU. Em todos os SAR-PDU's (exceto possivelmente no último), o SAR-PDU *payload* conterá 44 octetos de informação do CPCS-PDU.

Em cada SAR-PDU, o campo *LI* é ajustado como sendo o número de octetos de dados do SAR-SDU carregados em seu *payload*.

Quando o SAR transmissor estiver no estado IDLE, ele deve ajustar o bit mais significativo do campo *ST* do primeiro SAR-PDU para "1" ("BOM" ou "SSM"). Em todos os SAR-PDU's subseqüentes, este bit deve ser ajustado para "0" ("COM" ou "EOM").

Quando SAR transmissor receber uma primitiva *SAR_U_Abort_invoke* ele simplesmente a ignora.

II.4.3.1.2 Procedimentos da subcamada SAR do lado receptor

Será mostrado neste ítem todo comportamento do protocolo referente à subcamada SAR, quando se comportando como receptor. Partindo de sua máquina de estados, será observado quais decisões devem ser tomadas pelo protocolo, quando do recebimento de determinadas primitivas. Abaixo estão, sua máquina de estados e uma tabela discriminando seus estados.

Estado	Definição
IDLE	Esperando para receber nova SAR-SDU
REASS	Recebendo uma SAR-SDU

Tabela II.4 - Definições dos estados de SAR receptor

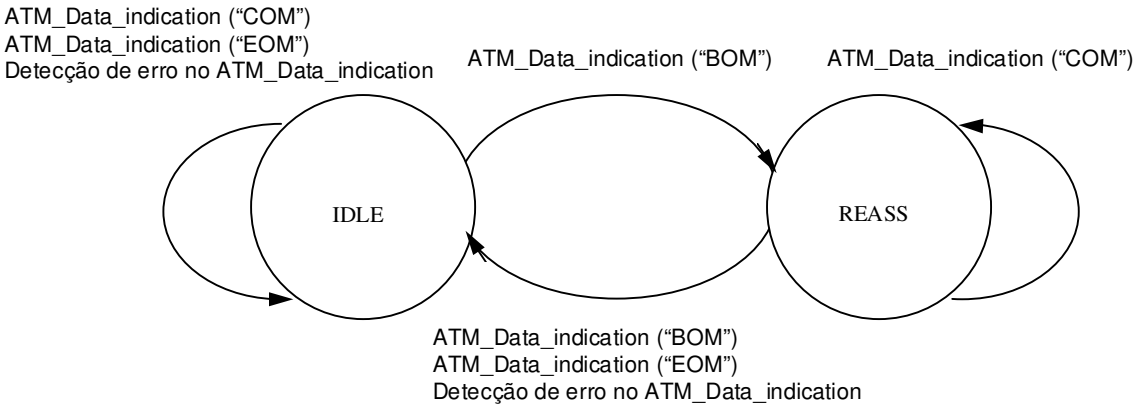


Figura II.7 - Máquina de estados de SAR receptor

A função principal da subcamada SAR receptor é a de ajudar na remontagem da mensagem que anteriormente foi segmentada para possibilitar sua transmissão pela rede ATM. Quando do recebimento das células transmitidas pelo ATM, esta subcamada executa a análise e "retirada" do cabeçalho e da cauda que foram inseridos quando de sua transmissão, e caso todos os SAR-PDU's estiverem corretos, sem perdas ou trocas, eles são repassados para a subcamada CPCS.

O SAR receptor faz uso de uma variável de estado auxiliar denominada *rcv_SN*, que tem a função de detectar perda ou ganho de SAR-PDU's. Depois do recebimento de um SAR-PDU com o campo *ST* indicando "COM" ou "EOM", o SAR receptor compara o valor contido no campo *SN* com esta variável. Se forem iguais, o SAR-PDU é assumido como estando dentro da seqüência e a variável *rcv_SN* é incrementada de um (módulo 16). Se não forem iguais é sinal que houve troca na ordem de transmissão dos SAR-PDU's.

Para todo SAR-PDU recebido, o SAR receptor verifica se o valor do campo *LI* está dentro de seus valores permitidos de acordo com o campo *ST* (Tabela II.1). Se o valor de *LI* não atender a este requisito, o SAR-PDU é descartado resultando também no envio de uma primitiva *SAR_P_Abort_signal* para a subcamada CPCS.

Na ausência de erros e independentemente do estado no qual o SAR receptor estiver, o número de octetos indicados no campo *LI* são enviados do SAR-PDU *payload* para o CPCS. Se o campo *ST* indica "EOM" ou "SSM", o parâmetro *M* é ajustado em "0" e o SAR receptor procederá ao estado IDLE. Se o *ST* indica "BOM" ou "COM", o parâmetro *M* é ajustado em "1" e o SAR receptor procederá para ou permanecerá no estado REASS.

Se o SAR receptor estiver no estado IDLE e receber um SAR-PDU cujo campo *ST* indica "COM" ou "EOM", ele ignorará este SAR-PDU.

Se o SAR receptor estiver no estado REASS e receber um SAR-PDU cujo campo *ST* indica "BOM" ou "SSM", isso resultará no envio de uma primitiva *SAR_P_Abort_signal* ao CPCS.

Se o SAR receptor estiver no estado REASS e receber um SAR-PDU cujo valor do campo *SN* não for igual ao valor da variável de estado *rcv_SN*, isso resultará no envio de uma primitiva *SAR_P_Abort_signal* ao CPCS. Também, se o campo *ST* indicar "COM" ou "EOM", o SAR-PDU será descartado e o SAR receptor procederá ao estado IDLE.

II.4.3.2 Procedimentos da subcamada CPCS

II.4.3.2.1 Procedimentos da subcamada CPCS do lado transmissor

Será mostrado neste ítem todo comportamento do protocolo referente à subcamada CPCS, quando se comportando como transmissor. Partindo de sua máquina de estados, será observado quais decisões devem ser tomadas pelo protocolo, quando do recebimento de determinadas primitivas. Abaixo estão, sua máquina de estados e uma tabela discriminando seus estados.

Estado	Definição
IDLE	Esperando para transmitir nova CPCS-SDU

Tabela II.5 - Definições dos estados de CPCS transmissor

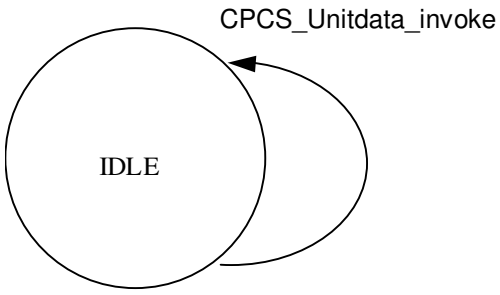


Figura II.8 - Máquina de estados de CPCS transmissor

A função principal da subcamada CPCS transmissor é a de receber o conjunto de dados vindos do SAP externo do AAL, e adicionar a estes dados um cabeçalho e uma cauda que servirão para permitir posterior verificação da validade dos dados transportados pela rede ATM.

O CPCS transmissor faz uso de uma variável de estado auxiliar denominada *snd_BEtag* que é usada para ajustar os valores dos campos *Btag* no cabeçalho e *Etag* na cauda do CPCS-PDU.

Quando do estabelecimento da conexão, o CPCS transmissor ajustarará a variável *snd_BEtag* com algum valor entre 0 e 255.

Quando do recebimento de uma primitiva *CPCS_Unitdata_invoke* do usuário, o CPCS transmissor construirá o cabeçalho, alocará o CPCS-SDU recebido dentro do CPCS-PDU *payload*, alocará, se necessário o *PAD* e construirá a cauda. A partir disso, o CPCS-PDU é então remetido inteiramente à subcamada SAR através de uma primitiva *SAR_Unitdata_invoke* (com parâmetro *M* igual a zero) para segmentação e transmissão.

Após o envio do CPCS-PDU para o SAR, o CPCS transmissor modificará a variável de estado *snd_BEtag*.

II.4.3.2.2 Procedimentos da subcamada CPCS do lado receptor

Será mostrado neste ítem todo comportamento do protocolo referente à subcamada CPCS, quando se comportando como receptor. Partindo de sua máquina de estados, será observado quais decisões devem ser tomadas pelo protocolo, quando do recebimento de determinadas primitivas. Abaixo estão, sua máquina de estados e uma tabela discriminando seus estados.

Estado	Definição
IDLE	Esperando para começar a remontar nova CPCS-PDU
REASS	Remontando uma CPCS-PDU
ABORT	Abortando uma CPCS-PDU ilegal

Tabela II.6 - Definições dos estados de CPCS receptor

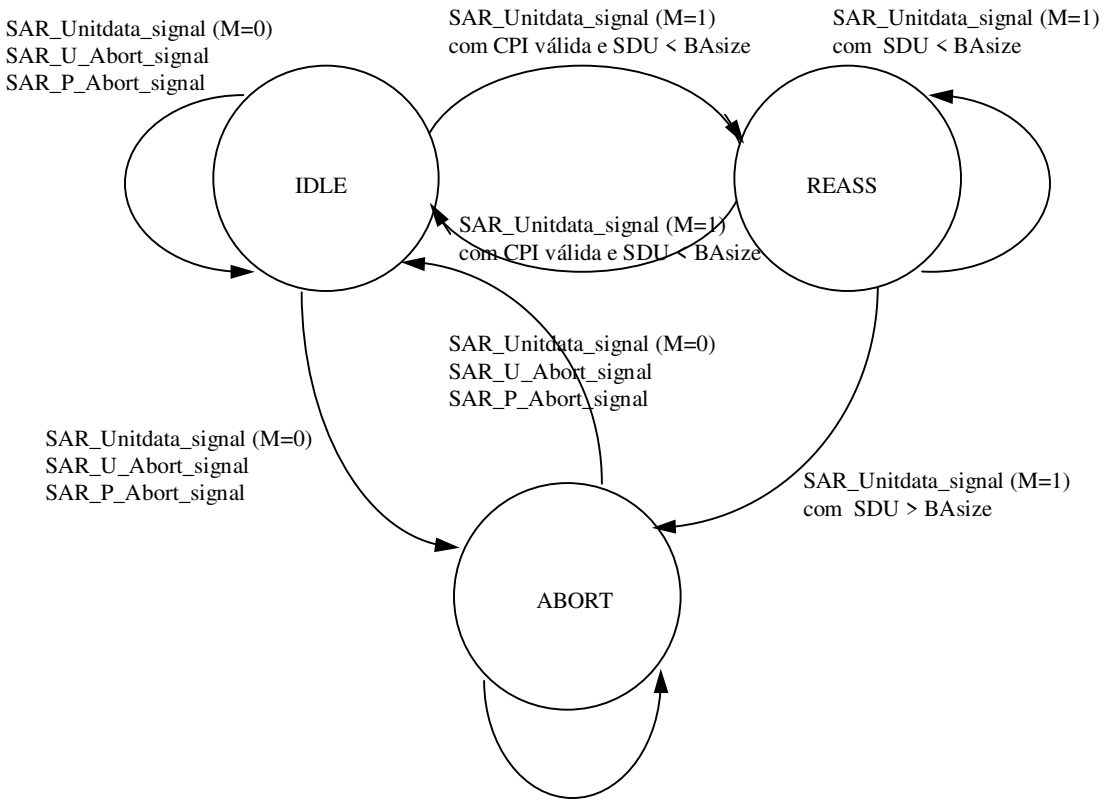


Figura II.9 - Máquina de estado de CPCS receptor

A função principal do CPCS receptor é a de remontar a mensagem anteriormente segmentada com ajuda da subcamada SAR. O CPCS receptor aloca um espaço suficiente para remontagem da mensagem. A cada SAR-PDU recebido da subcamada SAR, o CPCS insere este SAR-PDU no espaço reservado para remontagem. Após a conclusão desta operação, o CPCS receptor analisa e "retira" seu cabeçalho e sua cauda e remete a mensagem original enviada pelo usuário distante ao usuário do AAL pelo SAP externo via primitiva *CPCS_Unitdata_signal*.

O CPCS receptor faz uso de duas variáveis de estado auxiliares denominadas *rcv_BEtag* e *rcv_BAsize*, a primeira é usada para assegurar que a cauda recebida pertença ao CPCS-PDU que está sendo remontado, e a segunda é usada para assegurar que as tentativas de se montar um CPCS-PDU maiores que o original falharão.

Quando o CPCS receptor está no estado IDLE e recebe uma primitiva *SAR_Unitdata_signal* da subcamada SAR, os primeiros quatro octetos de informação correspondem ao cabeçalho do CPCS-PDU. Se o campo *CPI* contido no cabeçalho for ilegal, o CPCS receptor procederá ao estado ABORT se o parâmetro *M* for "1" ou procederá para o estado IDLE se o parâmetro *M* for "0". Se *CPI* for legal, CPCS receptor copiará o

valor do campo *Btag* para a variável de estado *rcv_BEtag*. E também ajustará o valor da outra variável de estado *rcv_BAsize* ao valor do campo *BAsize*. A alocação de um espaço (ou *buffer*) para remontagem com o tamanho indicado pela variável *rcv_BAsize* é a parte principal desta operação.

Quando o CPCS receptor está no estado IDLE ou no estado REASS e recebe uma primitiva *SAR_Unitdata_signal* da subcamada SAR com o parâmetro *M* igual a "0", os últimos quatro octetos de informação representam a cauda do CPCS-PDU.

O CPCS receptor verificará se o valor do campo *Etag* é igual ao valor da variável *rcv_BEtag*, se não forem iguais, CPCS receptor limpará o *buffer* de remontagem e procederá ou permanecerá no estado IDLE.

Após receber o último SAR-PDU, o CPCS receptor copiará a informação contida no *buffer* de remontagem (CPCS-SDU remontado, sem o cabeçalho e sem a cauda) para o usuário AAL através do parâmetro *ID* da primitiva *CPCS_Unitdata_signal*. A quantidade de informação contida neste parâmetro *ID* é igual ao valor do campo *Lenght* da cauda do CPCS-PDU. Feito isto, CPCS receptor limpará o *buffer* de remontagem e procederá ou permanecerá no estado IDLE.

Quando o CPCS receptor está no estado IDLE ou no estado REASS e recebe uma primitiva *SAR_Unitdata_signal* da subcamada SAR com o parâmetro *M* igual a "1", não há presença da cauda do CPCS-PDU.

Se o CPCS receptor recebe uma primitiva *SAR_U_Abort_signal* ou uma primitiva *SAR_P_Abort_signal* da subcamada SAR quando estiver no estado IDLE, estas primitivas são ignoradas, se estiver no estado REASS, CPCS receptor limpará o *buffer* de remontagem e procederá ao estado IDLE.

Quando o CPCS receptor está no estado ABORT e recebe uma primitiva *SAR_Unitdata_signal* com o parâmetro *M* igual a "1", a primitiva é ignorada e CPCS receptor continuará no estado ABORT. Entretanto, CPCS receptor procederá ao estado IDLE se estando do estado ABORT, receber uma primitiva *SAR_U_Abort_signal* ou uma primitiva *SAR_P_Abort_signal* ou ainda uma primitiva *SAR_Unitdata_signal* com o parâmetro *M* igual a "0".

II.5 Especificação em Linguagem Estelle

Com o crescente aumento, tanto em redes de computadores quanto em automação de processos, do uso de comunicação de dados, surgiu a necessidade da criação de técnicas de descrição formal de protocolos, de modo a obter-se uma especificação clara, concisa e sem ambigüidades e que também permitisse a verificação, a simulação e a obtenção de uma implementação automática a partir dessa especificação.

Neste trabalho foi escolhido dentre as Técnicas de Especificação Formal (FDT - *Formal description Techniques*) adotadas internacionalmente pela ISO, a linguagem Estelle (*Extended State Transition Language*), que é um formalismo usado para descrição formal de sistemas distribuidos e, em particular, de protocolos de comunicação. Esta escolha se deve ao fato da facilidade de modelagem de máquinas de estados finita e também por haver disponibilidade de ferramentas para Estelle no ambiente do GTA (Grupo de Teleinformática e Automação) da UFRJ.

II.5.1 Arquitetura da Especificação

A arquitetura adotada para especificação é mostrada na Figura II.11. Esta arquitetura modela a troca de dados entre dois usuários da Camada de Adaptação ATM (AAL) (UA e UB). A troca de dados, bem como todos os procedimentos adotados contra eventuais extravios de informações e verificação dessas informações, é controlada pelas entidades AAL_A e AAL_B, conectadas através do meio de comunicação ATM-network.

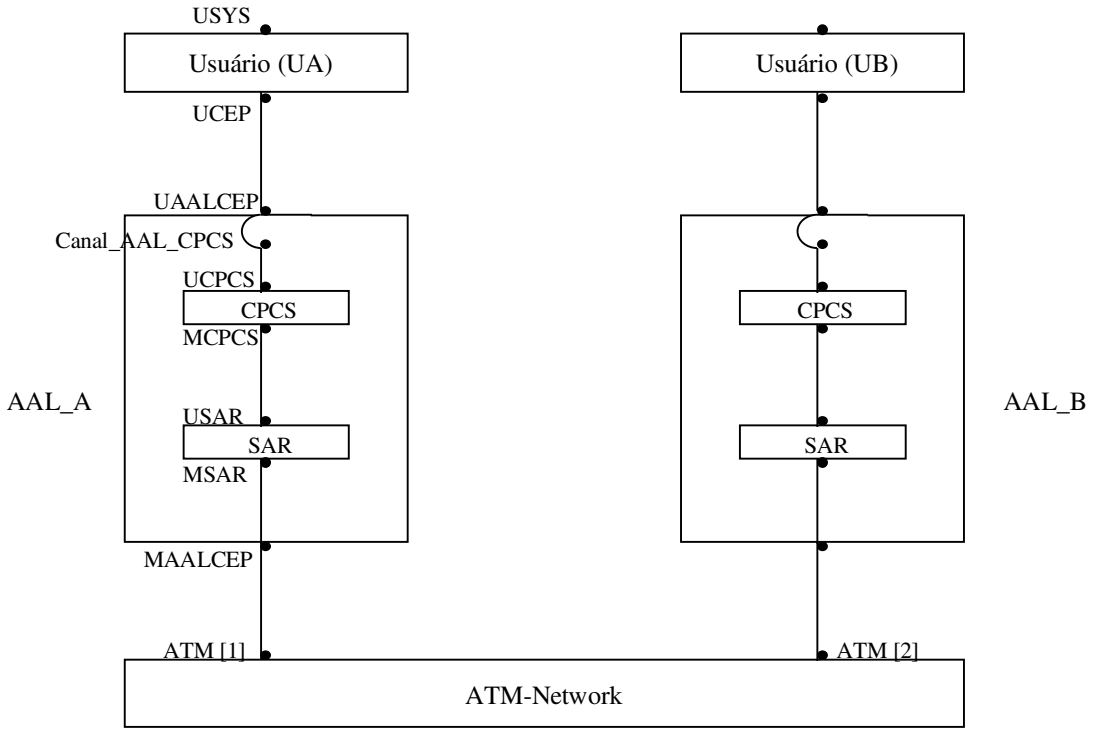


Figura II.10 - Arquitetura da especificação do protocolo AAL

II.5.2 Descrição da Especificação

Neste trabalho foi definido um único módulo Pai (AAL_type) dentro do qual foram definidos módulos filhos (CPCS_type e SAR_type) e também foi definido um meio de comunicação (ATM_network_type) e um módulo usuário (User_type).

No módulo principal foram declaradas constantes e tipos (PDU's, formato e estrutura), além da declaração de funções de escrita e leitura de teclado, e outras de manipulação de strings definidas como primitivas através do uso da diretiva {$ esternal NetUser}, que possibilita o uso dessas funções definidas externamente.

Os formatos definidos nas PDU's seguem expressamente a recomendação da ITU-T, e foi mostrado no ítem II.4.2. A única alteração em relação à recomendação é a adoção para todos os campos do cabeçalho e da cauda o formato de *array of char*, visto que a linguagem Estelle não possui a manipulação de bits e por isso não se poderia expressar esses campos conforme a recomendação (manipulando bits).

Em seguida são declarados os tipos de canais utilizados:

- canal *medium_interface* - usado para conexão entre o módulo Pai (AAL_type) com o meio de comunicação (ATM-network) e também para conexão entre o módulo filho SAR_type com o SAP externo do módulo Pai;
- canal *AAL_interface* - usado para conexão entre o módulo Pai (AAL_type) com o módulo usuário (User_type);
- canal *AAL_CPCS_interface* - usado para conexão entre o módulo filho CPCS_type com o ponto de interação interno do módulo Pai (AAL_type);
- canal *CPCS_SAR_interface* - usado para conexão entre o módulo filho CPCS_type com o outro módulo filho SAR_type.

Para melhor visualização, a Figura II.12 representa a arquitetura da especificação com os módulos e seus canais de interação.

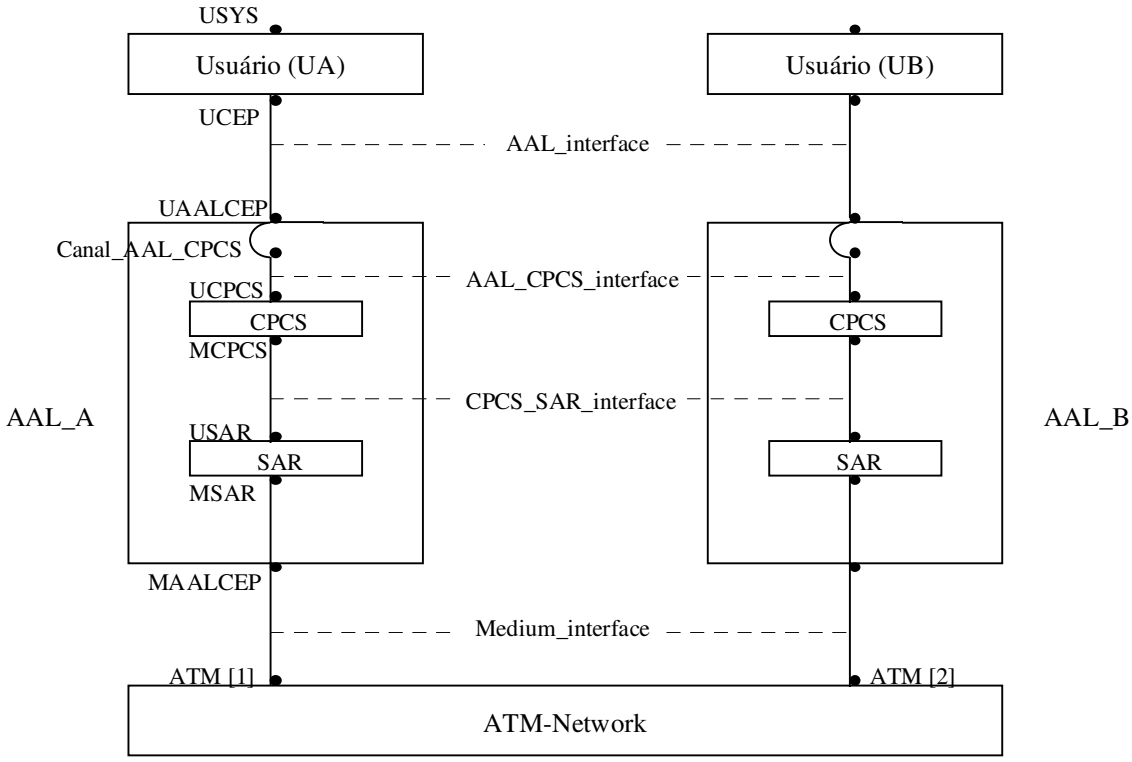


Figura II.11 - Arquitetura da especificação com módulos e canais de interação

Então declara-se o cabeçalho do módulo usado na definição da entidade que presta o serviço, e em seguida, declara-se o corpo desta mesma entidade. O corpo de módulo da entidade que presta o serviço proposto possui a especificação, em Estelle, da máquina de estados do serviço (vista em II.4.3), sendo a parte principal da especificação. Nele são

declaradas as variáveis locais, os estados existentes, as *procedures* e *functions* usadas no programa e, por fim, as transições.

A especificação completa em Estelle deste trabalho se encontra no Apêndice A.

A especificação em Estelle desse trabalho foi realizada em ambiente Sun na UFRJ. Partindo das máquinas de estado existentes na recomendação, foi-se definindo, após os canais de interação, módulos e corpos de módulos, as transições necessárias para realização do serviço proposto.

II.6 Comentários

Este capítulo apresentou uma descrição detalhada do protocolo escolhido para implementação e também uma descrição do serviço proposto por ele.

Inicialmente foi feita uma apresentação das primitivas utilizadas pelo protocolo para prestar o serviço proposto. Em seguida, uma descrição das funções e da codificação utilizada pelas subcamadas que compõem o protocolo. Por fim, foram apresentados os procedimentos adotados pelo protocolo para realização de sua finalidade, e foram apresentadas as máquinas de estado das subcamadas e comentado em linguagem simples esses procedimentos.

Neste capítulo também foi apresentada a especificação formal do protocolo, baseada nas Técnicas de Descrição Formal da ISO, em linguagem Estelle, de modo a tornar sua implementação clara e concisa.

O protocolo apresentado neste trabalho teve sua máquina de estados totalmente especificada e simulada em Estelle, com o auxílio das ferramentas do CAD ESTELLE desenvolvidas pelo GTA da UFRJ.

Introduziu-se ao Gerador Automático de Seqüências de Testes a máquina de estado referente ao serviço proposto. Como resultado, foi gerado um relatório com diversas seqüências de envio/recebimento de primitivas de serviço. Os resultados desta geração de seqüências e também comentários a seu respeito serão mostrados no próximo capítulo.

Introduziu-se no Simulador do CAD Estelle, as seqüências de envio/recebimento de primitivas de serviço oriundas do Gerador Automático de Seqüências de Testes. Os resultados referentes a esta operação e também comentários a seu respeito serão mostrados no próximo capítulo.

CAPÍTULO III
Resultados

III.1 Introdução

A geração de seqüências de testes e a simulação são ferramentas utilizadas para se verificar a funcionalidade da especificação de um determinado protocolo. Devido a sua importância, será feita uma descrição das seqüências de testes geradas pelo programa GASTPC e comentários a seu respeito. E também será mostrada a simulação a qual o protocolo foi submetido, bem como comentários a seu respeito.

III.2 Geração Automática de Seqüências de Testes

Introduziu-se ao Gerador Automático de Seqüências de Testes a máquina de estados referente ao serviço proposto. Esta máquina de estados abrange todas as outras máquinas de estados mostradas anteriormente, referentes às subcamadas SAR e CPCS (ítens II.4.3.1 e II.4.3.2). A máquina de estados do serviço representa o protocolo como um todo, como se houvesse uma "caixa preta" contendo as subcamadas SAR e CPCS, especificadas através de suas máquinas de estados.

A máquina de estados referente ao serviço do protocolo é mostrada na figura III.1. Esta máquina é mais simples que as das subcamadas SAR e CPCS, pois os procedimentos mais complexos como inserção e retirada de cabeçalhos e caudas, e segmentação e remontagem de mensagens, são efetuadas dentro das subcamadas.

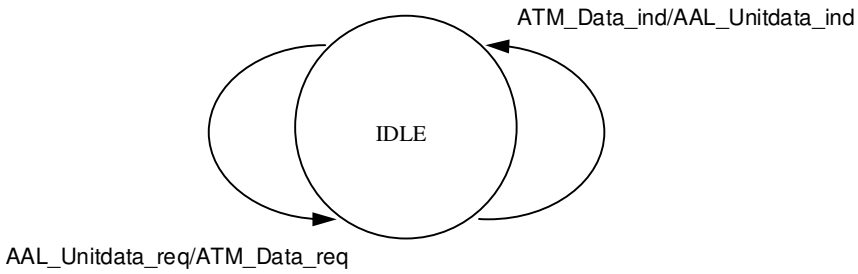


Figura III.1 - Máquina de estados do serviço AAL

Como resultado do programa GASTPC, foi gerado um relatório com diversas seqüências de envio/recebimento de primitivas de serviço. Estas seqüências abrangem todas as possíveis transições que a máquina de estados deve efetuar para realizar de maneira eficaz o serviço proposto.

A figura III.2 mostra as seqüências geradas pelo programa GASTPC.

```
r/N,AAL_Unitdata_req/ATM_Data_req,AAL_Unitdata_req/ATM_Data_req
r/N,ATM_Data_ind/AAL_Unitdata_ind,AAL_Unitdata_req/ATM_Data_req
r/N,AAL_Unitdata_req/N,AAL_Unitdata_req/ATM_Data_req
r/N,ATM_Data_ind/N,AAL_Unitdata_req/ATM_Data_req
r/N,r/N,AAL_Unitdata_req/ATM_Data_req
```

Figura III.2 - Seqüências de testes do serviço AAL

III.3 Simulação

O teste definitivo da funcionalidade do protocolo especificado é realizado através da inserção das seqüências de teste geradas pelo GASTPC no Simulador.

O Simulador possui duas funções especiais que permitem uma visualização total ou passo a passo do comportamento do protocolo. A função *random* gera seqüências de primitivas aleatoriamente durante um determinado número de passos especificados pelo usuário. A função *step* é utilizada para acompanhar passo a passo o comportamento do protocolo, através sa inserção de primitivas pelo usuário. É neste ponto que se utiliza as seqüências geradas pelo GASTPC.

Durante a fase do projeto do protocolo especificado neste trabalho, houve uma impossibilidade de se utilizar o Simulador do CAD Estelle. Devido a este fato, foi utilizado o Implementador Estelle que será Tese de Mestrado de Ildemar Pinto Nunes da COPPE.

O Implementador executa a compilação do protocolo especificadoem Estelle e gera um arquivo executável do protocolo. A partir daí, foram feitos testes da funcionalidade durante a execução do protocolo. Estes testes são mostrados no Apêndice B.

III.4 Comentários

Este capítulo mostrou a importância da Geração Automática de Seqüências de Testes em conjunto com a Simulação de modo a se verificar a funcionalidade da especificação em Estelle do protocolo apresentado neste trabalho.

Inicialmente foram mostradas as diversas seqüências de envio/recebimento de primitivas de serviço geradas pelo GASTPC.

Ainda neste capítulo foi mostrada a importância da utilização do Simulador a partir do relatório do GASTPC, e explicado o motivo da utilização do Implementador Estelle neste trabalho de modo a testar sua funcionalidade.

Foi observada a total funcionalidade da especificação realizada.

CAPÍTULO IV
Conclusões

Relembrando o que foi mencionado na Introdução deste trabalho, a importância da especificação deste protocolo é resultado do crescimento, no cenário mundial, da RDSI-BL e do ATM quando da integração de serviços existentes, e também no caso de surgimento de novas aplicações a alta velocidade.

Este trabalho apresentou o projeto de um protocolo AAL tipo 3/4 com as seguintes características:

- Transferência confiável de dados entre dois usuários que estejam utilizando como meio uma rede ATM;
- Prestação de serviços não orientados à conexão;
- Segmentação e remontagem da informação;
- Detecção de erros nos dados (mas sem retransmissão).

O protocolo foi implementado em linguagem formal (Estelle). Utilizando-se ferramentas de auxílio a desenvolvimento de protocolos, foi feita a implementação da especificação através do Implementador Estelle e verificada sua funcionalidade através de testes na transmissão de dados.

A implementação do protocolo foi executada em ambiente UNIX, em estações de trabalho SunSparc2.

Como não foi utilizado o Simulador com as seqüências de testes geradas pelo GASTPC, não foi possível testar de uma maneira totalmente completa o serviço especificado. Este passo seria utilizado para afirmar se a especificação foi completa e sem erros ou não. Mas utilizando-se de sua implementação, o sistema mostrou-se funcional durante os testes

executados quando da transmissão de mensagens de diversos tamanhos. Mas voltando a afirmar, um diagnóstico completo de sua funcionalidade só poderá ser feito quando da utilização do Simulador.

Referências Bibliográficas

[1] International Tellecomunication Union (ITU), "*B-ISDN ATM Adaptation Layer (AAL) Specification*", ITU-T Recommendation I.363, março de 93.

[2] G. S. Silva, "Geração Automática de Seqüências de Testes para Protocolo de Comunicações", Master's thesis, Universidade Federal do Rio de Janeiro, PEL/COPPE/UFRJ, 143 p., Rio de Janeiro, Abril de 1992.

[3] R. C. de Oliveira Jr, "Um Compilador para a Linguagem Estelle", Master's thesis, Universidade Federal do Rio de Janeiro, PEL/COPPE/UFRJ, 135 p., Rio de Janeiro, Agosto de 1991.

[4] P. R. O. Valim, "Um Simulador de Protocolos de Comunicação para a Linguagem Estelle", Master's thesis, Universidade Federal do Rio de Janeiro, PEL/COPPE/UFRJ, 130 p., Rio de Janeiro, Abril de 1990.

[5] J. - Y. Le Boudec, "The Assynchrounous Transfer Mode (ATM) : a tutorial", 1992.

[6] G. J. Holzmann, "Design and Validation of Computer Protocols", Prentice-Hall International, Inc., 1991.

[7] L. M. R. Tarouco, "Redes de Computadores: Locais e de Longa Distância", McGraw-Hill, São Paulo, 1986.

[8] P. Dembinski and S.Budkowski, "The Formal Description Technique Estelle - Specification Language Estelle", North-Holland, 1989.

Apêndice A
Especificação formal em Estelle do protocolo AAL tipo 3/4

A listagem a seguir contém a especificação em Estelle do protocolo apresentado neste trabalho. Esta é a especificação utilizada nos testes finais de implementação.

```
(* Trabalho de Projeto de Fim de Curso     *)
(* Inicio em 20 de dezembro de 1995        *)

specification AAL_protocol systemactivity;
   default individual queue;
   timescale seconds;

   {$ external NetUserUDP}

   type string = array [0..255] of char;

   function  KeyPressed : boolean;                                          primitive;
   function  ReadKey : char;                                                primitive;
   procedure WriteLn;                                                       primitive;
   procedure WriteString (s : string);                                      primitive;
   procedure ReadString (VAR s : string);                                   primitive;
   Procedure WriteInt (i : integer);                                        primitive;
   procedure ClrScr;                                                        primitive;
   procedure Copy(str : string; inx : integer; len : integer; VAR result : string); primitive;
   function Lenght(VAR str : string) : integer;                             primitive;
   procedure Delete(VAR str : string; inx : integer; len : integer);        primitive;
   procedure Concat(s1 : string; s2 : string; VAR dest : string);           primitive;


   const maxstring = 187;
   type
         string1 = array [0..maxstring] of char;
         string2 = array [0..191] of char;
         string3 = array [0..62] of char;
         bit2 = array [0..1] of char;
         bit4 = array [0..3] of char;
         bit6 = array [0..5] of char;
         bit10 = array [0..9] of char;
         byte1 = array [0..7] of char;
         byte2 = array [0..15] of char;
```

```
stringaux = array [0..255] of char;

bufferremontagem = string1;


SDU_type = string1;

PDU_type = string3;


CPCS_header_type = RECORD
                        CPI : byte1;
                        Btag : byte1;
                        BAsize : byte2;
                   END;


CPCS_trailer_type = RECORD
                        AL : byte1;
                        Etag : byte1;
                        Lenght : byte2;
                    END;


SAR_header_type = RECORD
                      ST : bit2;
                      SN : bit4;
                      MID : bit10;
                  END;


SAR_trailer_type = RECORD
                        LI : bit6;
                        CRC : bit10;
                    END;



CPCS_PDU_type = RECORD
                    header : CPCS_header_type;
                    Data_CPCS_PDU : string2;
                    trailer : CPCS_trailer_type;
                    M : 0..1;
                END;


SAR_PDU_type = RECORD
                    header : SAR_header_type;
                    Data_SAR_PDU : string3;
                    trailer : SAR_trailer_type;
                    M : 0..1;
                END;


SAR_Data_type = RECORD
                    Data_PDU : string3;
                    M : 0..1;
                END;
```

```
channel medium_interface (user, provider);
by user:
  ATM_Data_req (SAR_PDU : SAR_PDU_type);
by provider:
  ATM_Data_ind (SAR_PDU : SAR_PDU_type);

channel AAL_interface (user, provider);
by user:
  AAL_Unitdata_req (SDU : SDU_type);
by provider:
  AAL_Unitdata_ind (SDU : SDU_type);

channel CPCS_SAR_interface (user, provider);
by user:
  SAR_Unitdata_invoke (CPCS_PDU : CPCS_PDU_type);
  SAR_U_Abort_invoke;
by provider:
  SAR_Unitdata_signal (SAR_Data : SAR_Data_type);
  SAR_U_Abort_signal;
  SAR_P_Abort_signal;

channel AAL_CPCS_interface (user, provider);
by user:
  CPCS_Unitdata_invoke (SDU : SDU_type);
by provider:
  CPCS_Unitdata_signal (SDU : SDU_type);

channel SYS_interface (user,provider);
by user:
        ENVIAR;


module ATM_network_type activity;
ip
        ATM : array [1..2] of medium_interface (provider);
end;

body ATM_network_body for ATM_network_type;

initialize
 name inicia_meio:
 begin
 end;

trans
    when ATM[1].ATM_Data_req
        name enviandoparasegunda:
        begin
            WriteString ('Transicao Envia PDU para a segunda estacao - Primitiva ATM_Data_ind');
            WriteLn;
            output ATM[2].ATM_Data_ind (SAR_PDU);
        end;
    when ATM[2].ATM_Data_req
        name enviandoparaprimeira:
```

```
          begin
             WriteString ('Transicao Envia PDU para a primeira estacao - Primitiva ATM_Data_ind');
                    WriteLn;
             output ATM[1].ATM_Data_ind (SAR_PDU);
          end;
  end; (* ATM_network body *)




module AAL_type activity;
  ip UAALCEP : AAL_interface (provider);
     MAALCEP : medium_interface (user);
end;

body AAL_body for AAL_type;
  ip  Canal_AAL_CPCS : AAL_CPCS_interface (user);
          Canal_SAR_AAL : medium_interface (provider);

 state  Idle;

module CPCS_type activity;
  ip UCPCS : AAL_CPCS_interface (provider);
     MCPCS : CPCS_SAR_interface (user);
end;

body CPCS_body for CPCS_type;

var
   snd_BEtag : integer;
   rcv_BEtag : integer;
   rcv_BAsize : integer;
   zerar : integer;
   tamanhobuffer : integer;
   len : integer;
   len2 : integer;
   inteiro1 : integer;
   inteiro2 : integer;
   sucess1 : boolean;
   sucess2 : boolean;
   rcv_BEtag_aux : byte1;
   rcv_BAsize_aux : byte2;
   rcv_AL_aux : byte1;
   rcv_CPI_aux : byte1;
   rcv_Etag_aux : byte1;
   rcv_Lenght_aux : byte2;
   binario1 : byte1;
   binario2 : byte2;
   SDU_aux : SDU_type;
   CPCS_PDU_aux : CPCS_PDU_type;
   buffer_aux : bufferremontagem;


state
    Idle, Reass, Abort;

procedure Inttobinario1 (snd_BEtag : integer; var binario1 : byte1);
          var index, i, k : integer;
```

```
    teste : byte1;
  begin
    i := snd_BEtag;
    if i = 0 then teste := '00000000'
            else begin
                    index := 7;
                    while i > 1 do
                            begin
                              k := i mod 2;
                              if k = 1 then teste[index] := '1'
                                      else teste[index] := '0';
                              i := i div 2;
                              index := index - 1;
                            end;
                    teste[index] := '1';
                    while index <> 0 do
                            begin
                              index := index - 1;
                              teste[index] := '0';
                            end;
                  end;
    binario1 := teste;
  end;

procedure Inttobinario2 (len : integer; var binario2 : byte2);
  var index, i, k : integer;
    teste : byte2;
  begin
    i := len;
    if i = 0 then teste := '0000000000000000'
            else begin
                    index := 15;
                    while i > 1 do
                            begin
                              k := i mod 2;
                              if k = 1 then teste[index] := '1'
                                      else teste[index] := '0';
                              i := i div 2;
                              index := index - 1;
                            end;
                    teste[index] := '1';
                    while index <> 0 do
                            begin
                              index := index - 1;
                              teste[index] := '0';
                            end;
                  end;
    binario2 := teste;
  end;

procedure Binariotoint1 (rcv_BEtag_aux : byte1; var inteiro1 : integer);
  var k, index, soma : integer;
    aux : array [0..0] of char;
    aux1 : char;
  begin
    soma := 0;
    k := 1;
    index := 7;
```

```
    while index <> 0 do
            begin
              aux1 := rcv_BEtag_aux [index];
              aux[0] := aux1;
              if aux = '1'
                    then soma := soma + k;
              k := 2 * k;
              index := index - 1;
            end;
    aux1 := rcv_BEtag_aux [0];
    aux[0] := aux1;
    if aux = '1' then soma := soma + k;
    inteiro1 := soma;
  end;


procedure Binariotoint2 (rcv_Lenght_aux : byte2; var inteiro2 : integer);
    var k, index, soma : integer;
       aux : array [0..0] of char;
       aux1 : char;
    begin
      soma := 0;
      k := 1;
      index := 15;
      while index <> 0 do
            begin
              aux1 := rcv_Lenght_aux [index];
              aux[0] := aux1;
              if aux = '1'
                    then soma := soma + k;
              k := 2 * k;
              index := index - 1;
            end;
      aux1 := rcv_Lenght_aux [0];
      aux[0] := aux1;
      if aux = '1' then soma := soma + k;
      inteiro2 := soma;
    end;


function VerificaCPI (SAR_Data : SAR_Data_type) : boolean;
    var aux : array [0..0] of char;
       aux1 : char;
    begin
      Copy (SAR_Data.Data_PDU,0,8,rcv_CPI_aux);
      if rcv_CPI_aux = '00000000'  then VerificaCPI := true
                                        else VerificaCPI := false;
      aux1 := rcv_CPI_aux [1];
      aux[0] := aux1;
      if (aux <> '1') and (aux <> '0') then VerificaCPI := true;
    end;


function VerificaMSAR (SAR_Data : SAR_Data_type) : boolean;
    begin
      if  SAR_Data.M = 0  then VerificaMSAR := true
                          else VerificaMSAR := false;
    end;
```

```
function VerificaAL (buffer_aux : bufferremontagem) : boolean;
        var len3, x : integer;
        begin
          len3 := Lenght (buffer_aux);
          x := len3 - 32;
          Copy (buffer_aux,x,8,rcv_AL_aux);
          if rcv_AL_aux = '00000000'   then VerificaAL := true
                                       else VerificaAL := false;
        end;


procedure  CriaCPCS_PDU (SDU : SDU_type; var CPCS_PDU_aux : CPCS_PDU_type);
        var i : integer;
        begin
          CPCS_PDU_aux.header.CPI := '00000000';
          Inttobinario1 (snd_BEtag,binario1);
          CPCS_PDU_aux.header.Btag := binario1;
          Inttobinario2 (len,binario2);
          CPCS_PDU_aux.header.BAsize := binario2;
          Copy (SDU,0,len,CPCS_PDU_aux.Data_CPCS_PDU);
          i := 3 - ((len + 3) mod 4);
          while i <> 0 do
                begin
                  CPCS_PDU_aux.Data_CPCS_PDU[len] := '0';
                  len := len + 1;
                  i := i - 1;
                end;
          CPCS_PDU_aux.trailer.AL := '00000000';
          CPCS_PDU_aux.trailer.Etag := binario1;
          CPCS_PDU_aux.trailer.Lenght := binario2;
          CPCS_PDU_aux.M := 0;
        end;


procedure TrataheaderCPCS (SAR_Data : SAR_Data_type);
        begin
          Copy (SAR_Data.Data_PDU,8,8,rcv_BEtag_aux);
          Copy (SAR_Data.Data_PDU,16,16,rcv_BAsize_aux);
          Binariotoint1 (rcv_BEtag_aux,inteiro1);
          Binariotoint2 (rcv_BAsize_aux,inteiro2);
          rcv_BEtag := inteiro1;
          rcv_BAsize := inteiro2;
        end;


procedure TratatrailerCPCS (buffer_aux : bufferremontagem; var sucess1 : boolean; var sucess2 : boolean;
var rcv_BAsize : integer);
        var len3, x, y, Etag, Lenght1 : integer;
        begin
          if VerificaAL(buffer_aux)
                then begin
                        len3 := Lenght (buffer_aux);
                        x := len3 - 24;
                        Copy (buffer_aux,x,8,rcv_Etag_aux);
                        x := len3 - 16;
                        Copy (buffer_aux,x,16,rcv_Lenght_aux);
```

```
                        Binariotoint1 (rcv_Etag_aux,inteiro1);
                        Binariotoint2 (rcv_Lenght_aux,inteiro2);
                        Etag := inteiro1;
                        Lenght1 := inteiro2;
                    end;
      if rcv_BEtag = Etag  then sucess1 := true
                                else sucess1 := false;
      if rcv_BAsize = Lenght1 then sucess2 := true
                                 else sucess2 := false;
    end;




initialize to Idle
    begin
      snd_BEtag := 10;
      sucess1 := false;
      sucess2 := false;
      rcv_BEtag := 0;
      rcv_BAsize := 0;
      tamanhobuffer := 0;
    end;

trans
    from Idle to Idle
        when UCPCS.CPCS_Unitdata_invoke
        name A1:
        begin
          WriteString('Transicao A1 - Envia primitiva SAR_Unitdata_invoke');
          WriteLn;
          len := Lenght (SDU);
          CriaCPCS_PDU (SDU,CPCS_PDU_aux);
          snd_BEtag := snd_BEtag + 1;
          output MCPCS.SAR_Unitdata_invoke (CPCS_PDU_aux);
        end;

    from Idle to Idle
        when MCPCS.SAR_U_Abort_signal
        name A2:
        begin
          WriteString('Transicao A2');
          WriteLn;
        end;

    from Idle to Idle
        when MCPCS.SAR_P_Abort_signal
        name A3:
        begin
          WriteString('Transicao A3');
          WriteLn;
        end;

    from Idle
        when MCPCS.SAR_Unitdata_signal
        provided VerificaMSAR (SAR_Data)
        to same
        name A4:
```

```
            begin
              WriteString('Transicao A4');
              WriteLn;
            end;

            provided NOT(VerificaMSAR(SAR_Data)) and NOT(VerificaCPI(SAR_Data))
            to Abort
            name A5:
            begin
               WriteString('Transicao A5');
               WriteLn;
            end;

            provided NOT(VerificaMSAR(SAR_Data)) and (VerificaCPI(SAR_Data))
            to Reass
            name A6:
            begin
           WriteString('Transicao A6 - Comeca remontar informacao');
              WriteLn;
              TrataheaderCPCS (SAR_Data);
              SDU_aux := SAR_Data.Data_PDU;
              tamanhobuffer := rcv_BAsize;
              Copy (SDU_aux,0,63,buffer_aux);
            end;

from Abort to Idle
            when MCPCS.SAR_U_Abort_signal
            name A7:
            begin
               WriteString('Transicao A7');
               WriteLn;
            end;

from Abort to Idle
            when MCPCS.SAR_P_Abort_signal
            name A8:
            begin
               WriteString('Transicao A8');
               WriteLn;
            end;

from Abort
            when MCPCS.SAR_Unitdata_signal
            provided VerificaMSAR(SAR_Data)
            to Idle
            name A9:
            begin
               WriteString('Transicao A9');
               WriteLn;
            end;

            provided NOT (VerificaMSAR(SAR_Data))
            to same
            name A10:
            begin
               WriteString('Transicao A10');
               WriteLn;
            end;
```

```
from Reass to Idle
        when MCPCS.SAR_U_Abort_signal
        name A11:
        begin
          WriteString('Transicao A11');
          WriteLn;
          zerar := 0;
          for zerar := 0 to tamanhobuffer do
                begin
                   buffer_aux[zerar] := '0';
                   zerar := zerar + 1;
                end;
        end;

from Reass to Idle
        when MCPCS.SAR_P_Abort_signal
        name A12:
        begin
           WriteString('Transicao A12');
           WriteLn;
           zerar := 0;
           for zerar := 0 to tamanhobuffer do
                begin
                   buffer_aux[zerar] := '0';
                   zerar := zerar + 1;
                end;
        end;

from Reass
        when MCPCS.SAR_Unitdata_signal
        provided VerificaMSAR(SAR_Data)
        to Idle
        name A13:
        begin
          WriteString('Transicao A13 - Envia primitiva CPCS_Unitdata_signal');
          WriteLn;
          Concat (buffer_aux,SAR_Data.Data_PDU,buffer_aux);
          TratatrailerCPCS (buffer_aux, sucess1, sucess2, rcv_BAsize);
          if (VerificaAL (buffer_aux)) and (sucess1) and (sucess2)
                then begin
                        len2 := Lenght (buffer_aux);
                        len2 := len2 - 32;
                        Delete (buffer_aux,len2,32);
                        Delete (buffer_aux,0,32);
                        len2 := Lenght (buffer_aux);
                        len2 := len2 - rcv_BAsize;
                        Delete (buffer_aux,rcv_BAsize,len2);
                        output UCPCS.CPCS_Unitdata_signal (buffer_aux);
                        zerar := 0;
                        for zerar := 0 to tamanhobuffer do
                                begin
                                  buffer_aux[zerar] := '0';
                                  zerar := zerar + 1;
                                end;
                    end
                else begin
                         zerar := 0;
```

```
                                    for zerar := 0 to tamanhobuffer do
                                        begin
                                          buffer_aux[zerar] := '0';
                                          zerar := zerar + 1;
                                        end;
                                end;
                        end;

                        provided NOT(VerificaMSAR(SAR_Data))
                        to same
                        name A14:
                        begin
                          WriteString('Transicao A14 - Remonta informacao');
                          WriteLn;
                          SDU_aux := SAR_Data.Data_PDU;
                          Concat (buffer_aux,SDU_aux,buffer_aux);
                        end;

end; (*CPCS_body*)


module SAR_type activity;
  ip USAR : CPCS_SAR_interface (provider);
     MSAR : mediuminterface (user);
end;

body SAR_body for SAR_type;

type codigo_ST_type = (BOM,COM,EOM);

var
   snd_SN : integer;
   count : integer;
   countaux : integer;
   SN : integer;
   rcv_SN : integer;
   len : integer;
   LI : integer;
   M : integer;
   inteiro3 : integer;
   inteiro4 : integer;
   sucess : boolean;
   snd_ST : bit2;
   LI_aux : bit6;
   SN_aux : bit4;
   binario3 : bit4;
   binario4 : bit6;
   SAR_PDU_aux : SAR_PDU_type;
   SAR_Data_aux : SAR_Data_type;
   Prim_aux : stringaux;
   Seg_aux : stringaux;
   Terc_aux : stringaux;
   Total_aux : stringaux;


state
          Idle, Reass;
```

```
procedure Inttobinario3 (snd_SN : integer; var binario3 : bit4);
        var index, i, k : integer;
           teste : bit4;
        begin
          i := snd_SN;
          if i = 0 then teste := '0000'
                    else begin
                              index := 3;
                              while i > 1 do
                                        begin
                                          k := i mod 2;
                                          if k = 1 then teste[index] := '1'
                                                    else teste[index] := '0';
                                          i := i div 2;
                                          index := index - 1;
                                        end;
                              teste[index] := '1';
                              while index <> 0 do
                                        begin
                                          index := index - 1;
                                          teste[index] := '0';
                                        end;
                         end;
          binario3 := teste;
        end;

procedure Inttobinario4 (len : integer; var binario4 : bit6);
        var index, i, k : integer;
           teste : bit6;
        begin
          i := len;
          if i = 0 then teste := '000000'
                    else begin
                              index := 5;
                              while i > 1 do
                                        begin
                                          k := i mod 2;
                                          if k = 1 then teste[index] := '1'
                                                    else teste[index] := '0';
                                          i := i div 2;
                                          index := index - 1;
                                        end;
                              teste[index] := '1';
                              while index <> 0 do
                                        begin
                                          index := index - 1;
                                          teste[index] := '0';
                                        end;
                         end;
          binario4 := teste;
        end;

procedure Binariotoint3 (LI_aux : bit6; var inteiro3 : integer);
        var k, index, soma : integer;
           aux : array [0..0] of char;
           aux1 : char;
        begin
          soma := 0;
```

```
        k := 1;
        index := 5;
        while index <> 0 do
                begin
                  aux1 := LI_aux [index];
                  aux[0] := aux1;
                  if aux = '1'
                          then soma := soma + k;
                  k := 2 * k;
                  index := index - 1;
                end;
        aux1 := LI_aux [0];
        aux[0] := aux1;
        if aux = '1' then soma := soma + k;
        inteiro3 := soma;
      end;


  procedure Binariotoint4 (SN_aux : bit4; var inteiro4 : integer);
        var k, index, soma : integer;
           aux : array [0..0] of char;
           aux1 : char;
        begin
          soma := 0;
          k := 1;
          index := 3;
          while index <> 0 do
                  begin
                    aux1 := SN_aux [index];
                    aux[0] := aux1;
                    if aux = '1'
                            then soma := soma + k;
                    k := 2 * k;
                    index := index - 1;
                  end;
          aux1 := SN_aux [0];
          aux[0] := aux1;
          if aux = '1' then soma := soma + k;
          inteiro4 := soma;
        end;

  procedure  CriaATM_Data_req (var Total_aux : stringaux; var SAR_PDU_aux : SAR_PDU_type; count :
integer;  M : integer);
        var i : integer;
        begin
          SAR_PDU_aux.header.ST := snd_ST;
          Inttobinario3 (snd_SN,binario3);
          SAR_PDU_aux.header.SN := binario3;
          SAR_PDU_aux.header.MID := '0000000000';
          if count >= 63 then begin
                                  Copy (Total_aux,0,63,SAR_PDU_aux.Data_SAR_PDU);
                                  Delete (Total_aux,0,63);
                                  len := 63;
                              end
                          else begin
                                  countaux := 63 - count;
                                  i := countaux;
                                  len := count;
```

```
                                    Copy (Total_aux,0,count,SAR_PDU_aux.Data_SAR_PDU);
                                    while i <> 0 do
                                      begin
                                            SAR_PDU_aux.Data_SAR_PDU [count] := '0';
                                            count := count + 1;
                                            i := i - 1;
                                      end;
                                end;
          Inttobinario4 (len,binario4);
          SAR_PDU_aux.trailer.LI := binario4;
          SAR_PDU_aux.trailer.CRC := '0000000000';
          SAR_PDU_aux.M := M;
          snd_SN := snd_SN + 1;
        end;



function VerificaMCPCS (CPCS_PDU : CPCS_PDU_type) : boolean;
        begin
          if CPCS_PDU.M = 0 then VerificaMCPCS := true
                            else VerificaMCPCS := false;
        end;


function VerificaST (SAR_PDU : SAR_PDU_type; var sucess : boolean; var LI : integer) : codigo_ST_type;
        var teste : bit2;
        begin
          teste := SAR_PDU.header.ST;
          LI_aux := SAR_PDU.trailer.LI;
          Binariotoint3 (LI_aux,inteiro3);
          LI := inteiro3;
          if ((teste = '10') and (LI = 63)) then begin
                                                   sucess := true;
                                                   VerificaST := BOM;
                                                 end;
          if ((teste = '00') and (LI = 63)) then begin
                                                   sucess := true;
                                                   VerificaST := COM;
                                                 end;
          if (teste = '01')
                then if ((LI >= 2) and (LI <= 63))
                        then begin
                                sucess := true;
                                VerificaST := EOM;
                             end;
        end;


function VerificaMID (SAR_PDU : SAR_PDU_type) : boolean;
        begin
          if SAR_PDU.header.MID = '0000000000' then VerificaMID := true
                                               else VerificaMID := false;
        end;


function VerificaCRC (SAR_PDU : SAR_PDU_type) : boolean;
        begin
          if SAR_PDU.trailer.CRC = '0000000000' then VerificaCRC := true
```

```
                                                    else VerificaCRC := false;
        end;


initialize to Idle
        begin
          snd_SN := 1;
          sucess := false;
          LI := 0;
          SN := 0;
          rcv_SN := 0;
        end;


trans
        from Idle
                when USAR.SAR_Unitdata_invoke
                provided VerificaMCPCS (CPCS_PDU)
                to Idle
                name B1:
                begin
                  WriteString('Transicao B1 - Envia primitiva ATM_Data_req');
                  WriteLn;
                  Concat (CPCS_PDU.header.CPI,CPCS_PDU.header.Btag,Prim_aux);
                  Concat (CPCS_PDU.header.BAsize,CPCS_PDU.Data_CPCS_PDU,Seg_aux);
                  Concat (CPCS_PDU.trailer.AL,CPCS_PDU.trailer.Etag,Terc_aux);
                  Concat (Prim_aux,Seg_aux,Total_aux);
                  Concat (Total_aux,Terc_aux,Total_aux);
                  Concat (Total_aux,CPCS_PDU.trailer.Lenght,Total_aux);
                  snd_ST := '10';
                  len := Lenght (Total_aux);
                  count := len;
                  repeat
                          M := 1;
                          CriaATM_Data_req (Total_aux, SAR_PDU_aux, count, M);
                          output MSAR.ATM_Data_req (SAR_PDU_aux);
                          snd_ST := '00';
                          count := count - 63;
                  until count <= 63;
                  if count = 0
                          then begin
                              end
                          else begin
                                  snd_ST := '01';
                                  M := 0;
                                  CriaATM_Data_req (Total_aux, SAR_PDU_aux, count, M);
                              end;
                  output MSAR.ATM_Data_req (SAR_PDU_aux);
                end;

        from Idle to Idle
                when USAR.SAR_U_Abort_invoke
                name B2:
                begin
                  WriteString('Transicao B2');
                  WriteLn;
                  snd_SN := 2;
                end;
```

```
        from Idle
                when MSAR.ATM_Data_ind
                provided ((VerificaST(SAR_PDU, sucess, LI) = EOM) and (sucess))
                to same
                name B3:
                begin
                  WriteString('Transicao B3');
                  WriteLn;
                end;

                provided ((VerificaST(SAR_PDU, sucess, LI) = COM) and (sucess))
                to same
                name B4:
                begin
                  WriteString('Transicao B4');
                  WriteLn;
                end;

                provided  NOT(sucess)
                to same
                name B5:
                begin
                  WriteString('Transicao B5');
                  WriteLn;
                  output USAR.SAR_P_Abort_signal;
                end;

                provided ( NOT(VerificaMID(SAR_PDU)) or NOT(VerificaCRC(SAR_PDU)) )
(* ERRO *)
                to same
                name B6:
                begin
                  WriteString('Transicao B6');
                  WriteLn;
                  output USAR.SAR_P_Abort_signal;
                end;

                provided ((VerificaST(SAR_PDU, sucess, LI) = BOM) and (sucess))
                to Reass
                name B7:
                begin
                  WriteString('Transicao B7 - Envia primitiva SAR_Unitdata_signal');
                  WriteLn;
                  SN_aux := SAR_PDU.header.SN;
                  Binariotoint4 (SN_aux,inteiro4);
                  SN := inteiro4;
                  rcv_SN := SN + 1;
                  SAR_Data_aux.Data_PDU := SAR_PDU.Data_SAR_PDU;
                  SAR_Data_aux.M := 1;
                  output USAR.SAR_Unitdata_signal (SAR_Data_aux);
                end;

        from Reass
                when MSAR.ATM_Data_ind
                provided ((VerificaST(SAR_PDU, sucess, LI) = COM) and (sucess))
                to same
                name B8:
```

```
                begin
                  SN_aux := SAR_PDU.header.SN;
                  Binariotoint4 (SN_aux,inteiro4);
                  SN := inteiro4;
                  if rcv_SN = SN
                        then begin
                                rcv_SN := SN + 1;
                                SAR_Data_aux.Data_PDU := SAR_PDU.Data_SAR_PDU;
                                SAR_Data_aux.M := 1;
                                WriteString('Transicao B8 - Envia primitiva SAR_Unitdata_signal');
                                WriteLn;
                                output USAR.SAR_Unitdata_signal (SAR_Data_aux);
                           end
                        else begin
                                WriteString('Transicao B8 - Envia primitiva SAR_P_Abort_signal');
                                WriteLn;
                                output USAR.SAR_P_Abort_signal;
                           end;
                end;

                provided ((VerificaST(SAR_PDU, sucess, LI) = EOM) and (sucess))
                to Idle
                name B9:
                begin
                  SN_aux := SAR_PDU.header.SN;
                  Binariotoint4 (SN_aux,inteiro4);
                  SN := inteiro4;
                  if rcv_SN = SN
                    then begin
                           rcv_SN := SN + 1;
                           Copy (SAR_PDU.Data_SAR_PDU,0,LI,SAR_Data_aux.Data_PDU);
                           SAR_Data_aux.M := 0;
                           WriteString('Processo concluido - Informacao remontada');
                           WriteLn;
                           WriteString('Transicao B9 - Envia primitiva SAR_Unitdata_signal');
                           WriteLn;
                           output USAR.SAR_Unitdata_signal (SAR_Data_aux);
                         end
                    else begin
                           WriteString('Transicao B9 - Envia primitiva SAR_P_Abort_signal');
                           WriteLn;
                           output USAR.SAR_P_Abort_signal;
                         end;
                end;

                provided ((VerificaST(SAR_PDU, sucess, LI) = BOM) and (sucess))
                to Idle
                name B10:
                begin
                  WriteString('Transicao B10 - Envia primitivas SAR_P_Abort_signal e
SAR_Unitdata_signal');
                  WriteLn;
                  output USAR.SAR_P_Abort_signal;
                  SN_aux := SAR_PDU.header.SN;
                  Binariotoint4 (SN_aux,inteiro4);
                  SN := inteiro4;
                  rcv_SN := SN + 1;
                  SAR_Data_aux.Data_PDU := SAR_PDU.Data_SAR_PDU;
```

```
                SAR_Data_aux.M := 1;
                output USAR.SAR_Unitdata_signal (SAR_Data_aux);
              end;

              provided NOT(sucess)
              to Idle
              name B11:
              begin
                WriteString('Transicao B11 - Envia primitiva SAR_P_Abort_signal');
                WriteLn;
                output USAR.SAR_P_Abort_signal;
              end;

              provided ( NOT(VerificaMID(SAR_PDU)) or NOT(VerificaCRC(SAR_PDU)) )
              to same
              name B12:
              begin
                WriteString('Transicao B12 - Envia primitiva SAR_P_Abort_signal');
                WriteLn;
                output USAR.SAR_P_Abort_signal;
              end;
        end;(*SAR_body*)

modvar
        CPCS : CPCS_type;
        SAR  : SAR_type;

initialize to Idle
        begin
          init CPCS with CPCS_body;
          init SAR  with SAR_body;
          connect CPCS.MCPCS to SAR.USAR;
          connect Canal_SAR_AAL to SAR.MSAR;
          connect Canal_AAL_CPCS to CPCS.UCPCS;
        end;

trans
        from Idle to Idle
        when UAALCEP.AAL_Unitdata_req
        name C1:
        begin
          WriteString('Transicao C1 - Envia primitiva CPCS_Unitdata_invoke');
          WriteLn;
          output Canal_AAL_CPCS.CPCS_Unitdata_invoke (SDU);
        end;

        from Idle to Idle
        when Canal_AAL_CPCS.CPCS_Unitdata_signal
        name C2:
        begin
          WriteString('Transicao C2 - Envia primitiva AAL_Unitdata_ind');
          WriteLn;
          output UAALCEP.AAL_Unitdata_ind (SDU);
        end;

        from Idle to Idle
        when MAALCEP.ATM_Data_ind
        name C3:
```

```
          begin
            WriteString('Transicao C3 - Envia primitiva ATM_Data_ind');
            WriteLn;
            output Canal_SAR_AAL.ATM_Data_ind (SAR_PDU);
          end;

          from Idle to Idle
          when Canal_SAR_AAL.ATM_Data_req
          name C4:
          begin
            WriteString('Transicao C4 - Envia primitiva ATM_Data_req');
            WriteLn;
            output MAALCEP.ATM_Data_req (SAR_PDU);
          end;
end;(*AAL_body*)



module User_type activity;
  ip UCEP : AAL_interface(user);
     USYS : SYS_interface(provider);
end;

body User_body for User_type;
  ip SYS  : SYS_interface(user);
     USER : SYS_interface(provider);

  state
          S1;

initialize to S1
          name inicia_usuario:
          begin
            connect SYS to USER;
            WriteString ('   *** Protocolo AAL Implementado em Estelle ***');
        WriteLn;
            WriteLn;
            WriteString ('       Comandos Disponiveis: (E)nviar');
            WriteLn;
            WriteLn;
          end;

trans
    provided KeyPressed
      var ch : array [0..0] of char;
          ch1: char;
    name LeTecla :
      begin
        ch1 := ReadKey;
        ch[0] := ch1;
        if (ch = 'e') or (ch = 'E') then output SYS.ENVIAR;
            WriteLn;
      end;

trans
          from S1 to S1
          when USER.ENVIAR
            var SDU_aux : SDU_type;
```

```
        name D1:
        begin
          WriteString ( 'Entre com a mensagem a enviar : ');
          ReadString ( SDU_aux );
          WriteLn;
          WriteString('Transicao D1 - Envia primitiva AAL_Unitdata_req');
          WriteLn;
          output UCEP.AAL_Unitdata_req (SDU_aux);
        end;

        from S1 to S1
        when UCEP.AAL_Unitdata_ind
        name D2:
        begin
          WriteString('Transicao D2 - Recebe primitiva AAL_Unitdata_ind');
          WriteLn;
          WriteString ( 'Chegou a mensagem : ');
          WriteString (SDU);
          WriteLn;
          WriteString ('       Comandos Disponiveis: (E)nviar');
          WriteLn;
        end;
end; (* User_body*)



  modvar
      AAL_A, AAL_B   : AAL_type;
      MEIO           : ATM_network_type;
      UA, UB         : User_type;

  initialize (* AAL_protocol *)
         name iniciaprotocolo:
         begin
            init MEIO    with ATM_network_body;
            init AAL_A   with AAL_body;
            init AAL_B   with AAL_body;
                    init UA      with User_body;
            init UB      with User_body;
            connect AAL_A.MAALCEP to MEIO.ATM[1];
            connect AAL_B.MAALCEP to MEIO.ATM[2];
            connect UA.UCEP to AAL_A.UAALCEP;
            connect UB.UCEP to AAL_B.UAALCEP;
         end;
end. (* AAL_protocol *)
```

Apêndice B
Acompanhamento de uma sessão de testes

Neste apêndice é mostrado o acompanhamento de uma pequena sessão de testes do protocolo AAL. Este acompanhamento foi necessário devido à não utilização do simulador e foi feito com a inserção de rotinas de impressão na tela dos pontos onde eram efetuadas transições durante a execução do protocolo. Com isso, vemos quais transições foram disparadas durante a fase de transmissão de mensagens.

```
  *** Protocolo AAL Implementado em Estelle ***
     Comandos Disponiveis: (E)nviar
  *** Protocolo AAL Implementado em Estelle ***
     Comandos Disponiveis: (E)nviar

 Inicio da Execucao ...
E
Entre com a mensagem a enviar : Este e o primeiro teste do protocolo.
Transicao D1 - Envia primitiva AAL_Unitdata_req
Transicao C1 - Envia primitiva CPCS_Unitdata_invoke
Transicao A1 - Envia primitiva SAR_Unitdata_invoke
Transicao B1 - Envia primitiva ATM_Data_req
Transicao C4 - Envia primitiva ATM_Data_req
Transicao Envia PDU para a segunda estacao - Primitiva ATM_Data_ind
Transicao C4 - Envia primitiva ATM_Data_req
```

Transicao Envia PDU para a segunda estacao - Primitiva ATM_Data_ind

Transicao C3 - Envia primitiva ATM_Data_ind

Transicao C3 - Envia primitiva ATM_Data_ind

Transicao B7 - Envia primitiva SAR_Unitdata_signal

Transicao A6 - Comeca remontar informacao

Processo concluido - Informacao remontada

Transicao B9 - Envia primitiva SAR_Unitdata_signal

Transicao A13 - Envia primitiva CPCS_Unitdata_signal

Transicao C2 - Envia primitiva AAL_Unitdata_ind

Transicao D2 - Recebe primitiva AAL_Unitdata_ind

Chegou a mensagem : Este e o primeiro teste do protocolo.

Comandos Disponiveis: (E)nviar

E

Entre com a mensagem a enviar : O protocolo efetuou perfeitamente o primeiro teste, tentaremos nova transmissao.

Transicao D1 - Envia primitiva AAL_Unitdata_req

Transicao C1 - Envia primitiva CPCS_Unitdata_invoke

Transicao A1 - Envia primitiva SAR_Unitdata_invoke

Transicao B1 - Envia primitiva ATM_Data_req

Transicao C4 - Envia primitiva ATM_Data_req

Transicao Envia PDU para a primeira estacao - Primitiva ATM_Data_ind

Transicao C3 - Envia primitiva ATM_Data_ind

Transicao C4 - Envia primitiva ATM_Data_req

Transicao Envia PDU para a primeira estacao - Primitiva ATM_Data_ind

Transicao C3 - Envia primitiva ATM_Data_ind

Transicao C4 - Envia primitiva ATM_Data_req

Transicao Envia PDU para a primeira estacao - Primitiva ATM_Data_ind

Transicao C3 - Envia primitiva ATM_Data_ind

Transicao B7 - Envia primitiva SAR_Unitdata_signal

Transicao A6 - Comeca remontar informacao

Transicao B8 - Envia primitiva SAR_Unitdata_signal

Transicao A14 - Remonta informacao

Processo concluido - Informacao remontada

Transicao B9 - Envia primitiva SAR_Unitdata_signal

Transicao A13 - Envia primitiva CPCS_Unitdata_signal

Transicao C2 - Envia primitiva AAL_Unitdata_ind

Transicao D2 - Recebe primitiva AAL_Unitdata_ind

Chegou a mensagem : O protocolo efetuou perfeitamente o primeiro teste, tentaremos nova transmissao.

Comandos Disponiveis: (E)nviar

E

Entre com a mensagem a enviar : Pelo visto, o protocolo executa perfeitamente o servico proposto, o de transmitir mensagens de um lado a outro da conexao.

Transicao D1 - Envia primitiva AAL_Unitdata_req

Transicao C1 - Envia primitiva CPCS_Unitdata_invoke

Transicao A1 - Envia primitiva SAR_Unitdata_invoke

Transicao B1 - Envia primitiva ATM_Data_req

Transicao C4 - Envia primitiva ATM_Data_req

Transicao Envia PDU para a segunda estacao - Primitiva ATM_Data_ind

Transicao C4 - Envia primitiva ATM_Data_req

Transicao Envia PDU para a segunda estacao - Primitiva ATM_Data_ind

Transicao C4 - Envia primitiva ATM_Data_req

Transicao Envia PDU para a segunda estacao - Primitiva ATM_Data_ind

Transicao C3 - Envia primitiva ATM_Data_ind

Transicao C3 - Envia primitiva ATM_Data_ind

Transicao C3 - Envia primitiva ATM_Data_ind

Transicao B7 - Envia primitiva SAR_Unitdata_signal

Transicao A6 - Comeca remontar informacao

Transicao B8 - Envia primitiva SAR_Unitdata_signal

Transicao A14 - Remonta informacao

Processo concluido - Informacao remontada

Transicao B9 - Envia primitiva SAR_Unitdata_signal

Transicao A13 - Envia primitiva CPCS_Unitdata_signal

Transicao C2 - Envia primitiva AAL_Unitdata_ind

Transicao D2 - Recebe primitiva AAL_Unitdata_ind

Chegou a mensagem : Pelo visto, o protocolo executa perfeitamente o servico proposto, o de transmitir mensagens de um lado a outro da conexao.